MAINVIEW® AutoOPERATOR™ Advanced Automation Guide for REXX EXECs

Version 6.2

March 15, 2002

bmcsoftware

Restricted Rights Legend

BMC Software, Inc.
2101 CityWest Blvd.
Houston TX 77042-2827
USA

Contacting BMC Software

You can access the BMC Software Web site at **http://www.bmc.com**. From this Web site, you can obtain general information about the company, its products, special events, and career opportunities. For a complete list of all BMC Software offices and locations, go to **http://www.bmc.com/corporate/offices.html**.

USA and Canada

Address	BMC Software, Inc. 2101 CityWest Blvd. Houston TX 77042-2827
Telephone	713 918 8800 or 800 841 2031
Fax	713 918 8000

Outside USA and Canada

Telephone	(01) 713 918 8800
Fax	(01) 713 918 8000

Customer Support

You can obtain technical support by using the Support page on the BMC Software Web site or by contacting Customer Support by telephone or e-mail. To expedite your inquiry, please see "Before Contacting BMC Software," below.

Support Web Site

You can obtain technical support from BMC Software 24 hours a day, seven days a week by accessing the technical support Web site at **http://www.bmc.com/support.html**. From this site, you can

- read overviews about support services and programs that BMC Software offers
- find the most current information about BMC Software products
- search a database for problems similar to yours and possible solutions
- order or download product documentation
- report a problem or ask a question
- subscribe to receive e-mail notices when new product versions are released
- find worldwide BMC Software support center locations and contact information, including e-mail addresses, fax numbers, and telephone numbers

Support via Telephone or E-mail

In the USA and Canada, if you need technical support and do not have access to the Web, call 800 537 1813. Outside the USA and Canada, please contact your local support center for assistance. To find telephone and e-mail contact information for the BMC Software support center that services your location, refer to the Contact Customer Support section of the Support page on the BMC Software Web site at **www.bmc.com/support.html**.

Before Contacting BMC Software

Before you contact BMC Software, have the following information available so that a technical support analyst can begin working on your problem immediately:

- product information
  - product name
  - product version (release number)
  - license number and password (trial or permanent)
- operating-system and environment information
  - machine type
  - operating system type, version, and service pack or program temporary fix (PTF)
  - system hardware configuration
  - serial numbers
  - related software (database, application, and communication) including type, version, and service pack or PTF
- sequence of events leading to the problem
- commands and options that you used
- messages received (and the time and date that you received them)
  - product error messages
  - messages from the operating system, such as `file system full`
  - messages from related software

Contents

Tables

Figures

About This Book

The *MAINVIEW AutoOPERATOR Advanced Automation Guide for REXX EXECs* is for system programmers who need to perform advanced automation tasks in the data center.

Use this manual with the MAINVIEW AutoOPERATOR product (also referred to simply as AutoOPERATOR) to learn about:

- How you can use REXX EXECs with AutoOPERATOR to create EXECs that you can use to automate your environment, including:
  - How AutoOPERATOR processes parameters in EXECs
  - How to use variables and variable pools
  - How to control EXEC execution in AutoOPERATOR
  - How to perform some advanced tasks with EXECs across targets
  - How to debug your AutoOPERATOR EXECs
- How to use the Open Systems Procedural Interface (OSPI) to interact with VTAM-based products

This manual also documents:

- The IMFEXEC command statements you can use with AutoOPERATOR EXECs
- AutoOPERATOR-supplied utility EXECs

How This Manual Is Organized

The manual contains the following chapters:

- Chapter 1. Introduction to Using AutoOPERATOR and EXECs to Automate Your Environment

  Discusses how you can
  - Use REXX EXECs and AutoOPERATOR IMFEXEC commands to write automation tasks
  - Use variables to save data
  - Control EXEC execution once you schedule the EXEC
- Chapter 2. Using REXX Conventions and Syntax in AutoOPERATOR REXX EXECs

  Describes the conventions, syntax, and restrictions for writing REXX EXECs.
- Chapter 3. Passing Parameters to REXX EXECs in AutoOPERATOR

  Describes how AutoOPERATOR interprets and uses information passed to EXECs in positional parameters.
- Chapter 4. Using Variables in REXX EXECs

  Describes the different types of variables and their pools and how to manipulate the pools.
- Chapter 5. Controlling EXEC Execution

  Describes the different ways you can send an EXEC to run and how to control its execution.

- Chapter 6. Using Advanced Techniques with AutoOPERATOR EXECs

  Describes how you can send EXECs, messages, and ALERTS to different targets with EXECs.

- Chapter 7. Accessing DB2 from AutoOPERATOR

  Describes how you can access DB2 from AutoOPERATOR with REXX EXECs if you have the BMC Software RxD2/LINK product installed.

- Chapter 8. Interacting with VTAM Applications with OSPI

  Describes how to use Open Systems Procedural Interface (OSPI) to communicate with VTAM applications.

- Chapter 9. Performing Automation Using AOAnywhere

  Describes how to use the AOAnywhere EXEC syntax to perform automation from outside the AutoOPERATOR BBI-SS PAS.

- Chapter 10. Accessing Array Data with AutoOPERATOR EXECs

  Describes how to use IMFEXEC ARRAY commands to access data collected in arrays.

- Chapter 11. Using the MAINVIEW API

  Describes commands, functions and facilities that allow AutoOPERATOR users to access data available on the MAINVIEW Databus with AutoOPERATOR EXECs.

- Chapter 12. Using the IMFEXEC Statements

  Lists the IMFEXEC command statements you can use with REXX to write EXECs to accomplish advanced automation tasks.

- Chapter 13. Testing and Debugging EXECs Interactively

  Describes when and how to use the AutoOPERATOR EXEC Tester and provides examples of its features.

- Chapter 14. Using the AutoOPERATOR-Supplied Utility EXECs

  Lists the AutoOPERATOR-supplied utility EXECs available with AutoOPERATOR.

This manual also contains:

- An appendix for SYSPROG service EXECs
- A glossary
- An index

MAINVIEW AutoOPERATOR Product Library

MAINVIEW AutoOPERATOR is available with seven options:

- MAINVIEW AutoOPERATOR for OS/390
- MAINVIEW AutoOPERATOR for IMS
- MAINVIEW AutoOPERATOR for CICS
- MAINVIEW AutoOPERATOR Access NV
- MAINVIEW AutoOPERATOR TapeSHARE
- MAINVIEW AutoOPERATOR for MQSeries
- MAINVIEW AutoOPERATOR Elan Workstation

The base product and these options are documented in the following MAINVIEW AutoOPERATOR manuals:

- *MAINVIEW AutoOPERATOR Customization Guide*
- *MAINVIEW AutoOPERATOR Basic Automation Guide*
- *MAINVIEW AutoOPERATOR Advanced Automation Guide for CLIST EXECs*
- *MAINVIEW AutoOPERATOR Advanced Automation Guide for REXX EXECs*
- *MAINVIEW AutoOPERATOR Options User Guide*
- *MAINVIEW AutoOPERATOR for MQSeries Installation and User Guide*
- *MAINVIEW AutoOPERATOR Reference Summary*
- *MAINVIEW AutoOPERATOR Solutions Guide*

This manual also makes several references to the BMC Software Intercommunications (BBI) PAS, which provides subsystem communication in its own MVS address space. The BBI online environment is described in the

- *MAINVIEW Common Customization Guide*
- *MAINVIEW Administration Guide*
- *Using MAINVIEW*

Recommended Reading

There is no recommended reading.

Related Reading

The following lists the IBM documents that are referenced in this guide:

- *MVS/ESA Initialization and Tuning Guide*, GC28-1635
- *TSO Extensions Version 2: CLISTs*, SC38-1876
- *TSO Extensions Version 2: REXX User's Guide*, SC28-1882
- *TSO Extensions Version 2: REXX Reference*, SC28-1883
- *TSO Extensions Version 2: Customization*, SC28-1872
- *TSO Extensions Version 2: Command Reference*, SC28-1881
- *CICS Supplied Transactions*, SC33-1686-02
- *CICS Operations and Utilities Guide*, SC33-1685
- *Routing and Descriptor Codes*, GC28-1194
- *Routing and Descriptor Codes*, GC28-1666
- *Routing and Descriptor Codes*, GC28-1816
- *Supervisor Services and Macro Instructions*, GC28-1154

and the following BMC Software documents:

- *MAINVIEW Common Customization Guide*
- *MAINVIEW Administration Guide*
- *Using MAINVIEW*
- MAINVIEW Quick Reference
- OS/390 and z/OS Installer Giude
- Implementing Security for MAINVIEW Products
- MAINVIEW Alternate Access Implementation and User Guide
- MAINVIEW AlarmManager User Guide
- *RxD2/LINK™ User Guide and Reference*
- *MAINVIEW for CICS User Guide*

What the Conventions Are

The following syntax notation is used in this manual. Do not enter the special characters.

- Brackets, [ ], enclose optional parameters or keywords.
- Braces, { }, enclose a list of parameters; one must be chosen.
- A vertical line, |, separates alternative options; one can be chosen.
- An *italicized* or underlined parameter is the default.
- AN ITEM IN CAPITAL LETTERS must be entered exactly as shown.
- Items in lowercase letters are values you supply.

Chapter 1. Introduction to Using AutoOPERATOR and EXECs to Automate Your Environment

This manual documents how you can use REXX EXECs with AutoOPERATOR to perform automation tasks on your system. If you would like to write CLIST EXECs for AutoOPERATOR, use the *MAINVIEW AutoOPERATOR Advanced Automation Guide for CLIST EXECs* manual.

For complete information for writing REXX EXECs, refer to the IBM publications *TSO Extensions Version 2: REXX/MVS User's Guide* and *TSO Extensions Version 2: REXX/MVS Reference*.

This chapter briefly discusses REXX EXECs and how you can use them with AutoOPERATOR to create programs to automate your environment. This chapter introduces the following concepts:

- Using EXECs with AutoOPERATOR
- Choosing the EXEC language
- The seven different ways an EXEC can be scheduled
- Passing information to EXECs
- Controlling EXEC execution
- Using variables in EXECs

Overview

Basic automation tasks, such as reacting to messages, are provided through facilities such as the AutoOPERATOR Rule Processor application. More complex automation tasks, including interfaces to performance, scheduling, and network products, require programs that can be tailored to specific site needs. These programs, called AutoOPERATOR EXECs, are written by system programmers or operators using either the TSO CLIST or TSO REXX language.

AutoOPERATOR EXECs:

- Are IBM TSO CLISTs and REXX programs with special language extensions for CICS, IMS, and MVS management through the use of IMFEXEC commands

  For a list of REXX commands that AutoOPERATOR does **not** support, refer to "Restrictions in REXX EXECs" on page 21.

- Use the same logical expression and operator syntax as TSO CLISTs and REXX programs and provide many of the same TSO symbolic control variables, built-in functions, assignment statements, and conditional statements.

  These are described in this book in "Using REXX Conventions and Syntax in AutoOPERATOR REXX EXECs" on page 13 and in the IBM publication, *TSO Extensions Version 2: REXX/MVS User's Guide*.

- Are upward-compatible with TSO releases and versions.

Table 1 shows where you can find more information in this book.

Table 1. Finding Additional Information

To learn more about...	**See page...**
Using REXX syntax, conventions, and built-in functions	13
Passing parameters to EXECs in AutoOPERATOR	23
Using variables	49
Controlling EXEC execution	73
Using advanced techniques	87
Using the IMFEXEC statements in AutoOPERATOR REXX EXECs	237

Choosing the EXEC Language: REXX or CLIST

For each task, you can choose either REXX or CLIST to write your EXECs with. CLIST is a language which is familiar to many system programmers, but REXX is being acclaimed for its simplicity and power.

There are some performance considerations:

- REXX EXECs perform approximately 25% faster than CLIST EXECs.
- CLIST EXEC performance can be improved by:
  - Placing all comments on statements which do not include executable statements
  - Coding REXX=YES in the AAOEXP00 member of BBPARM
- Using VLF can reduce both CPU and I/O consumption

  Refer to the IBM publication *TSO/E Version 2 Customization Manual* for information on how to use VLF.

For a complete discussion about writing TSO CLISTs, refer to the IBM publication *TSO Extensions Version 2: CLISTS*. For a complete discussion about writing TSO REXX EXECs, refer to the IBM publications *TSO Extensions Version 2: REXX/MVS User's Guide* and *TSO Extensions Version 2: REXX/MVS Reference*.

If you want to create CLIST EXECs for AutoOPERATOR, refer to the BMC Software publication *MAINVIEW AutoOPERATOR Advanced Automation Guide for CLIST EXECs*.

Invoking AutoOPERATOR EXECs

A system programmer or operator can interactively create EXECs (consisting of a subset of REXX commands and IMFEXEC commands) by using standard edit procedures. The EXECs are then stored in the online SYSPROC DD (or SYSEXEC DD for REXX EXECs) for later execution.

These EXECs are powerful programs that execute in the AutoOPERATOR environment and interact with a target, thus enabling you to create robust automation procedures.

All EXECs can be initiated or invoked from the SYSPROC DD (and the SYSEXEC DD for REXX EXECs) in one of seven ways:

EXEC	How it is invoked
Rule-initiated	Scheduled when a message or command matches an enabled Rule that specifies the name of an EXEC to be invoked.
ALERT-initiated	Scheduled when you enter any value into the `RSP` field of the ALERT Detail Display for an ALERT which has an `E` in the `IND` field. The `E` indicates that there is a follow-up EXEC associated with the ALERT. For information regarding the ALERT Management Facility, refer to the chapter “ALERT Management Facility” in the *MAINVIEW AutoOPERATOR Basic Automation Guide*.
User-initiated	Scheduled when a user enters an EXEC name from a BBI-TS. COMMAND line with the command prefix % or 4, or is entered as a parameter of the MVS MODIFY command when it is issued against a BBI-SS PAS. For example, F SYSB,%EXECA where EXECA is the name of the EXEC to be scheduled. You can also schedule a user-initiated EXEC from the AutoOPERATOR EXEC Manager application. Refer to the *MAINVIEW AutoOPERATOR Basic Automation Guide* for more information.
Time-initiated	Scheduled when the AutoOPERATOR Timer Facility invokes the specified EXEC at times you specify. You can use the AutoOPERATOR Timer Facility to schedule EXECs or the AutoOPERATOR-supplied sample solution @TIMER. Refer to the *MAINVIEW AutoOPERATOR Basic Automation Guide* for more information about using these methods.
EXEC-initiated	Scheduled when one EXEC (for example, EXECABC) contains an IMFEXEC SELECT command statement that invokes a second EXEC (for example, EXECXYZ). EXECs scheduled in this way can execute either synchronously or asynchronously (refer to “Invoking EXECs Synchronously with IMFEXEC SELECT(EXEC) WAIT(YES)” on page 78).

Externally initiated	Scheduled from outside of AutoOPERATOR when the program IMFSUBEX is called from a job step, as a subroutine of a user program, from TSO, or from another AutoOPERATOR address space.
End-of-Memory–initiated	Scheduled at end-of-memory when an initiator, a TSO user, or a started task is terminated.

These EXECs, once they are invoked, perform their specified tasks on your system. Refer to Chapter 3, “Passing Parameters to REXX EXECs in AutoOPERATOR” on page 23 for a complete discussion.

Passing Information to REXX EXECs

For a REXX EXEC to perform its tasks, it must be able to receive and retain information about the system. This information is passed to EXECs through:

- Statements called ARG statements

  The first statement in an AutoOPERATOR REXX EXEC must state that this is a REXX EXEC. The next statement is usually the ARG statement and it is coded with positional parameters that take values from the input that schedules the EXEC and makes those values available to the EXEC itself.

  Chapter 3, “Passing Parameters to REXX EXECs in AutoOPERATOR” on page 23 contains examples of ARG statements and the information that gets passed to them depending on the way the EXEC is invoked.

- Variables in variable pools

  Variables reside in four categories of variable pools and they receive and retain information that the EXEC requires to complete its tasks.

  Chapter 4, “Using Variables in REXX EXECs” on page 49 contains a discussion about variables and variable pools.

The table on the following two pages summarizes the different possible values for the positional parameters on a ARG statement for the seven different EXEC types. The table shows up to 11 positional parameters but there can be more (up to 255 bytes).

Positional Parameters for the ARG Statement

Positional Parameter	Rule-initiated EXEC	ALERT-initiated EXEC	User-initiated EXEC	EXEC-initiated EXEC
1	Refer to page 29	EXEC name	EXEC name	EXEC name
2	Refer to page 29	Refer to page 31	First optional parameter	First optional parameter
3	Refer to page 29	Refer to page 31	Second optional parameter	Second optional parameter
4	Refer to page 29	Refer to page 31	Third optional parameter	Third optional parameter
5	Refer to page 29	Refer to page 31	Fourth optional parameter	Fourth optional parameter
6	Refer to page 29	Refer to page 31	Fifth optional parameter	Fifth optional parameter
7	Refer to page 29	Refer to page 31	Sixth optional parameter	Sixth optional parameter
8	Refer to page 29	Refer to page 31	Seventh optional parameter	Seventh optional parameter
9	Refer to page 29	Refer to page 31	Eighth optional parameter	Eighth optional parameter
10	Refer to page 29	Refer to page 31	Ninth optional parameter	Ninth optional parameter
11	Refer to page 29	Refer to page 31	Tenth optional parameter	Tenth optional parameter

Note: Each EXEC type is discussed separately in Chapter 3, “Passing Parameters to REXX EXECs in AutoOPERATOR” on page 23. Refer to that chapter for more detailed information, especially for ALERT-initiated EXECs and Rule-initiated EXECs.

Positional Parameters for the ARG Statement

Positional Parameter	Time-initiated EXEC	Externally initiated EXEC	End-of-Memory EXEC or IMFEOM
1	EXEC name	EXEC name	sp 1
2	Target name	First optional parameter	NORMAL or ABNORMAL
3	IMS ID - Used only for AutoOPERATOR for IMS option	Second optional parameter	N/A
4	BBI-SS PAS subsystem identifier	Third optional parameter	N/A
5	Current Gregorian date	Fourth optional parameter	N/A
6	Time the EXEC is scheduled	Fifth optional parameter	N/A
7	Day of the week	Sixth optional parameter	N/A
8	Current Julian date	Seventh optional parameter	N/A
9	Elapsed time of the active IMS/VS. Used only for MAINVIEW AutoOPERATOR for IMS.	Eighth optional parameter	N/A
10	The IMS/VS restart type. Used only for MAINVIEW AutoOPERATOR for IMS.	Ninth optional parameter	N/A
11	Number of times the EXEC has been invoked. Used only for MAINVIEW AutoOPERATOR for IMS.	Tenth optional parameter	N/A

Note: Each EXEC type is discussed separately in Chapter 3, “Passing Parameters to REXX EXECs in AutoOPERATOR” on page 23. Refer to that chapter for more detailed information, especially for time-initiated EXECs. See “End-of-Memory–Initiated REXX EXEC” on page 46 for information about End-of-Memory—initiated EXECs.

Controlling EXEC Execution

Each EXEC represents a unit of work that needs to be completed. Just as any system that handles requests to complete work, AutoOPERATOR provides scheduling facilities for EXECs. EXECS are queued for execution to either:

- The Normal queue
- The Priority queue

When an EXEC is scheduled to either the Normal or Priority queue, it waits for a server, called a *thread*, to become available.

You can control how an EXEC executes on the system by first specifying:

- How many threads you define for each queue
- Which queue you want to schedule the EXEC for
- Whether you want the EXECs to execute synchronously or asynchronously
- What time limits you specify for the queues

Once an EXEC is scheduled and running, you can also use certain BBI control commands to manually manipulate the progress of the EXEC. Chapter 5, “Controlling EXEC Execution” on page 73 contains discussions for all these items.

Using Variables in AutoOPERATOR EXECs

Complex EXECs must be able to do much more than issue commands and return control to their callers. An EXEC must be able to request information from AutoOPERATOR (and other products), compare the information, compare the time elapsed since the last observation, and effect changes that other EXECs or products carry out.

This type of logic requires the ability to save information, either temporarily or permanently, in a simple manner so that it can be accessed later by the same EXEC or other EXECs.

To retain this information for EXECs, AutoOPERATOR provides four kinds of variables and variable pools. For a complete discussion, see Chapter 4, “Using Variables in REXX EXECs” on page 49.

Variable Pool Name	Description
TSO variables	Exist for the life of the EXEC. This chapter lists: • AutoOPERATOR–supplied TSO variables • Modifiable TSO variables • Non-modifiable TSO variables
LOCAL variables	LOCAL variables are stored in a pool that can be accessed only by the current EXEC and other EXECs (using IMFEXEC SELECT WAIT(YES)). AutoOPERATOR passes information to an EXEC in this pool. It is also used by AOAnywhere when sharing variables with an invoking EXEC. The LOCAL variable pool is freed when the EXEC ends and its contents are lost.

<table>
<tr><th>Variable Pool Name</th><th colspan="2">Description</th></tr>
<tr><td rowspan="2">Two types of GLOBAL variable pools: SHARED and PROFILE</td><td colspan="2">Can be saved for later executions of the same EXEC or other EXECs.
The use of the expression “GLOBAL variables” in this book refers to both SHARED and PROFILE variables.</td></tr>
<tr><td>SHARED variables

SHARED variables are stored in a pool that is accessible to all EXECs in the BBI-SS PAS . They can be read, modified, created and deleted by any number of EXECs or Rules. Since EXECs can access them simultaneously, their access should be serialized (see IMFEXEC VENQ and VDEQ). These variables exist in storage beyond the life of the EXEC that created them.
AutoOPERATOR creates a number of SHARED variables that contain system-specific information. SHARED variables are accessible to the Rules Processor and remain in memory when the subsystem is terminated. However, they are lost across IPLs or when a subsystem is restarted with the VPOOL=RESET option.
This chapter lists the AutoOPERATOR-supplied variables.</td><td>PROFILE variables

PROFILE variables are similar to SHARED variables with the exception that they are persistent across IPLs and their contents are never lost unless explicitly deleted.

PROFILE variables are not accessable from Rules.</td></tr>
<tr><td colspan="3">Note: Variable names must be at least 1 and not more than 32 characters in length. The contents of any variable cannot exceed 256 characters.</td></tr>
</table>

Chapter 2. Using REXX Conventions and Syntax in AutoOPERATOR REXX EXECs

This chapter describes statements and variables you can use for a REXX EXEC. For more complete information about writing REXX EXECs in general, refer to the IBM manuals:

- *TSO Extensions Version 2: REXX/MVS User's Guide*
- *TSO Extensions Version 2: REXX/MVS Reference*

Using Expressions and Operators in REXX EXECs

All of the arithmetic, comparative and logical operators described in the IBM publication *TSO Extensions Version 2 REXX Reference* guide are valid in a REXX EXEC expression running within AutoOPERATOR. An expression combines variables, whole numbers, and character strings with operators. For example, the EXEC statement:

```
IF CMD = SUBSTR(Z1,1,1) THEN ...
```

uses the comparative operator = in an expression with the REXX IF conditional statement to compare the first character of the character string in the `Z1` symbol to the value in the `CMD` symbol.

The function `SUBSTR` is a built-in REXX function that replaces the function call with specific characters from a character string. The actual characters are selected by specifying a starting position and a length for the portion of the character string to be used.

In this example, `SUBSTR` is replaced with the first character of the character string substituted for the `Z1` symbol.

Using Control Statements in REXX EXECs

AutoOPERATOR EXECS support the following REXX control statements[1].

Statement	Description
CALL	Used to invoke a routine or control the trapping of certain conditions.
EXIT	Used to leave a program unconditionally.
ITERATE	Alters the flow of control within a repetitive DO loop.
LEAVE	Causes immediate exit from one or more repetitive DO loops.
RETURN	Used to return control (and possibly a result) from a REXX program or internal routine to the point of its invocation. **Note:** If the EXEC is invoked with the IMFEXEC SELECT EXEC(exec) WAIT(yes) statement, the RETURN control statement can be used only to return control from the REXX EXEC. Passing a value (RESULT) is not supported.
SELECT	Used to conditionally execute one of several alternative instructions or sets of instructions.
SIGNAL	Causes an abnormal change in the flow of control, or controls the trapping of certain conditions.

Using Assignment Statements in REXX EXECs

AutoOPERATOR EXECs support the following REXX assignment statements.[2]

Statement	Description
ARG	Used to retrieve argument strings passed to a program or internal routine and assign them to variables.
PARSE	Used to assign data to one or more variables.
PULL	Used to read a string from the queue (data stack) and assign it to a variable.
symbol = data	This assignment statement is the most common way of changing the value of a variable.

[1] The descriptions for these REXX control statements are from the IBM publication, *TSO Extensions Version 2: REXX/MVS Reference*, Chapter 3, “Keyword Instructions”.

[2] The descriptions for these REXX assignment statements are from the IBM publication, *TSO Extensions Version 2: REXX/MVS Reference*, Chapter 3, “Keyword Instructions”.

Using Conditional Statements in REXX EXECs

AutoOPERATOR EXECs support the following REXX conditional statements.

Statement	Description
DO-WHILE-END	Executes a set of related instructions only while specific condition exists.
DO-UNTIL-END	Executes a set of related instructions until a specific condition is met.
DO-TO-BY-FOR	Executes a set of related instructions using special keywords to control the loop. See the *TSO Extensions Version 2 REXX Reference* guide for more information on these keywords.
DO-FOREVER	Executes a set or related instructions until a specific instruction is issued to end the loop (for example, LEAVE or SIGNAL).
IF-THEN-ELSE	Used to conditionally execute an instruction or set of instructions depending on the evaluation of the expression.

Using Built-In Functions in REXX EXECs

AutoOPERATOR supports the following REXX built-in functions.[3] For additional information on syntax and parameters to pass to the function, see the *TSO Extensions Version 2 REXX Reference* guide.

Built-in Function	Description
ABBREV( )	Determines whether a character string is an abbreviation of another character string.
ABS( )	Returns the absolute value of a number.
ADDRESS( )	Returns the name of the environment to which host commands are currently being submitted.
ARG( )	Returns an argument string or information about the argument strings to a program or internal routine.
CENTER( ) or CENTRE( )	Returns a string centered according to specifications.
COMPARE( )	Determines if two strings are equal and returns 0 if so. If they are not equal, the character position at which they become not equal is returned.
CONDITION( )	Returns the condition information associated with the current trapped condition.
COPIES( )	Concatenates strings together and returns the concatenated string.
C2D( )	Character to decimal. Returns the decimal value of the binary representation of a string.
C2X( )	Character to hexadecimal. Converts a character string to its hexadecimal representation.
DATATYPE( )	Determines whether a string is numeric or character. Also determines whether a string is alphanumeric, binary, a mixed SBCS/DBCS string, a DBCS string, lowercase, mixed case, a number, a symbol, uppercase, a whole number, or a hexadecimal number.
DATE( )	Returns the local date in the format: `dd mon yyyy`
DELSTR( )	Deletes a substring from a character string.
DELWORD( )	Deletes a string from a group of character strings.
DIGITS( )	Returns the current setting of NUMERIC DIGITS.
D2X( )	Decimal to hexadecimal. Returns a string of hexadecimal characters that represent a decimal number.
ERRORTEXT( )	Returns the error text associated with a particular error message number.
EXTERNALS( )	Always returns a 0. This function is used under VM/SP.

[3] The descriptions for these REXX built-in functions are from the IBM publication, *TSO Extensions Version 2: REXX/MVS Reference*, Chapter 4, “Keyword Instructions”.

Built-in Function	Description
FIND( )	Searches for a phrase within a character string and returns the position of the first word of the phrase in the string.
FORM( )	Returns the current setting of NUMERIC FORM.
FORMAT( )	Rounds and formats a number.
FUZZ( )	Returns the current setting of NUMERIC FUZZ.
INDEX( )	Searches for a character string within another character string and returns either the starting position of the character string being searched for or 0.
INSERT( )	Inserts a character string into another character string.
JUSTIFY( )	Formats blank-delimited words by adding pad characters between words to justify both margins.
LASTPOS( )	Returns the position of the last occurrence of one string within another.
LEFT( )	Returns a string containing the leftmost characters of a string.
LENGTH( )	Returns the length of a string.
LINESIZE( )	For AutoOPERATOR, always returns '131'.
MAX( )	Returns the largest number from a list of specified numbers.
MIN( )	Returns the smallest number from a list of specified numbers.
OVERLAY( )	Overlays part or all of a string with a new string.
POS( )	Returns the position of one string within another.
QUEUED( )	Returns the number of lines remaining in the queue at the time when the function is invoked.
RANDOM( )	Returns a pseudo-random nonnegative whole number.
REVERSE( )	Returns a string, swapped end for end.
RIGHT( )	Returns a string containing the rightmost characters of a string.
SIGN( )	Returns a number that indicates the sign of a number.
SOURCELINE( )	Returns a source line in the current EXEC.
SPACE( )	Formats the blank-delimited words in a string with pad characters between each word.
STRIP( )	Removes leading and/or trailing characters from a string.
SUBSTR( )	Returns the substring of a string.
SUBWORD( )	Returns a substring of a string of words. The number of words returned is specified by a length parameter.
SYMBOL( )	Returns the state of a symbol (BAD, LIT, or VAR).
TIME( )	Returns the local time. By default, the time is returned in the 24-hour clock format (hh:mm:ss).

Built-in Function	Description
TRACE( )	Returns trace actions currently in effect.
TRANSLATE( )	Translates characters in a string to other characters, or reorders characters in a string.
TRUNC( )	Returns the integer part of a number and, optionally, the number of decimal places specified.
USERID( )	While running under AutoOPERATOR, by default will return the subsystem (SS) ID of AutoOPERATOR. If a value is coded for the PREFIX parameter in BBPARM member AAOEXP00, that will be the value returned.
VALUE( )	Returns the value of a specified symbol.
VERIFY( )	Verifies that a string is composed of a predefined set of characters and returns the position of the first character in the string that is not within the predefined set of characters.
WORD( )	Returns a blank-delimited word from a string.
WORDINDEX( )	Returns the position of the first character in a specified blank-delimited word in a specified string.
WORDLENGTH( )	Returns the length of a specified blank-delimited word in a specified string.
WORDPOS( )	Searches a specified string for the first occurrence of a specified sequence of blank-delimited words and returns the word number of the first word of the specified sequence of blank-delimited words found in the specified string.
WORDS( )	Returns the number of blank-delimited words in a specified string.
X2C( )	Converts a hexadecimal string to a character string.
X2D( )	Converts a hexadecimal string to decimal format.

In addition to these built-in functions, if you have the BMC Software product RxD2/LINK product installed, AutoOPERATOR also has access to the following REXX built-in functions.

Built-in Function	Description
CONVSTCK(tod)	Converts the 8-byte TOD clock into display format of YYYYDDD HHMMSSTH.
CTOD(tod)	Converts the 8-byte TOD clock time into display format of HHMMSSTH.
F2C(f)	Converts a floating point string to a character string.
GBLVAR	Creates and manages the global variable environment.
P2C(p)	Creates a packed decimal string to a character string.
UENV(hcename,pgm)	Identifies to REXX Host Command Environment (HCE) called hcename, such that pgm will receive control for ADDRESS hcename.
VARSPF( )	Converts a compound REXX variable to a simple ISPF dialog variable.
WAITSEC( )	Specifies the number of seconds to wait before continuing to process.

Using TSO/E Functions for REXX EXECs

AutoOPERATOR supports the following TSO/E REXX functions.[4] For additional information on syntax and parameters to pass to the function, see the *TSO Extensions Version 2 REXX Reference* guide.

Function	Description
LISTDSI( )	Sets several variables that describe a data set and returns a function code of 0, 4, or 16 that shows the completion code.
MSG( )	Returns the previous status of message issuing, which can be ON or OFF. It also allows you to turn message issuing on or off.
OUTTRAP( )	Returns the name of the variable in which trapped output is stored, or if trapping is not in effect, returns the word off. It also can be used to set trapping into effect.
PROMPT( )	Returns the previous setting of prompting for the EXEC, which will always be OFF when running under AutoOPERATOR.
STORAGE( )	Returns a specified number of bytes of data from a specified storage address. It also allows an EXEC to modify storage.
SYSDSN( )	Returns a message indicating whether a data set exists and is available for use.
SYSVAR( )	Sets variables that describe the current environment. The variable set depends upon the option used.

[4] The descriptions for these REXX built-in functions are from the IBM publication, *TSO Extensions Version 2: REXX/MVS Reference*, Chapter 4, “Keyword Instructions”.

Using TSO/E REXX Commands in REXX EXECs

AutoOPERATOR supports the following TSO/E REXX commands[5] if you specify the ADDRESS MVS command prior to issuing the command. For additional information on syntax and usage of the commands, see the *TSO Extensions Version 2 REXX Reference* guide.

Command	Description
DELSTACK	Deletes the most recently created data stack that was created by the NEWSTACK command, and all elements on it. If a new data stack was not created, DELSTACK removes all the elements from the original data stack.
DROPBUF	Deletes the most recently created data stack buffer that was created by the MAKEBUF command, and all elements on the data stack in the buffer. To remove a specific data stack buffer and all buffers created after it, issue the DROPBUF command with the number of the buffer.
EXECIO	Can be used to perform input and output operations to and from a data set, a stack, or a list of variables.
MAKEBUF	Creates a new buffer on the data stack. The MAKEBUF command can be issued from REXX EXECs that execute in both the TSO/E address space and non-TSO/E address spaces.
NEWSTACK	Creates a new data stack and hides or isolates the current data stack. Elements on the previous data stack cannot be accessed until a DELSTACK command is issued to delete the new data stack and any elements remaining in it.
QBUF	Queries the number of buffers that were created on the data stack with the MAKEBUF command.
QELEM	Queries the number of data stack elements that are in the most recently created data stack buffer.
QSTACK	Queries the number of data stacks in existence for an EXEC that is executing.
SUBCOM	Queries the existence of a specified host command environment.

[5] The descriptions for these TSO/E REXX commands are from the IBM publication, *TSO Extensions Version 2: REXX/MVS Reference*, Chapter 10, “TSO/E REXX Commands”.

Restrictions in REXX EXECs

AutoOPERATOR REXX EXECs do not support the following REXX language facilities.

- Immediate Commands:
  - HI - Halt Interpretation, HT - Halt Typing
  - RT - Resume Typing, TS - Trace Start
  - TE - Trace End

AutoOPERATOR REXX EXECs do not support the following REXX function:

- XRANGE( )

AutoOPERATOR REXX EXECs do not support using the TSO/E CALL or TSO/E Service Facility (IKJEFFTSR) to give control to an authorized program.

Chapter 3. Passing Parameters to REXX EXECs in AutoOPERATOR

This chapter describes:

- The four components of a REXX EXEC
- The differences in the ways parameters are passed based on how an AutoOPERATOR REXX EXEC is invoked

For information about CLIST EXECs and AutoOPERATOR, refer to *MAINVIEW AutoOPERATOR Advanced Automation Guide for CLIST EXECs*.

Understanding the Four Components of a REXX EXEC

This section briefly describes the four components of REXX EXECs. There are four steps to writing REXX EXECs:

- Defining the language

  All EXECs are assumed to be CLIST EXECs unless the first statement identifies the EXEC as a REXX EXEC. Refer to the IBM publication *TSO Extensions Version 2: REXX/MVS User's Guide* for a complete discussion.

- Passing data

  You must include a statement—called the ARG statement—that defines the input parameters to be used by the EXEC logic.

- Documenting the EXEC

  You can include comments, enclosed by /* and */, throughout the EXEC to describe the purpose of the EXEC statements

- Writing the logic

  A logic section that contains REXX EXEC statements and commands, and AutoOPERATOR IMFEXEC commands that perform user-defined automation tasks. Use the IMFEXEC commands to specify the automation actions and commands you want the EXECs to perform.

Each of these parts is described in the following sections.

Defining the Language

The TSO/E processor assumes that it is executing a CLIST EXEC unless the first statement it encounters (the PROC statement) defines the EXEC as a REXX EXEC. For example, if the first statement looks like:

```
/* REXX EXEC */
```

then the EXEC is processed as a REXX EXEC.

Passing Data

The REXX EXEC receives data to perform its task through the ARG statement. AutoOPERATOR uses these parameters to pass values to an EXEC when the EXEC is invoked.

The information passed through the ARG statements varies, depending on the way the EXEC is invoked. For example, an EXEC can be invoked by a Rule or by a user and the values passed to the EXEC for these two methods are different.

The ARG statement syntax is:

```
[UPPER] ARG [template]
```

where:

UPPER **Optional.**

Forces translation of any character string to uppercase. If UPPER is not specified, then no translation takes place.

ARG Instructs REXX to process the arguments passed to this REXX EXEC.

template Describes the rules to be used in parsing the input parameters. The template is a list of symbols separated by blanks and/or patterns.

Handling Strings of Periods

AutoOPERATOR passes all variables required by the type of EXEC **plus** a character string of ". . . . . ". The sum of the number of characters in this string and the number of characters in the variables passed to the EXEC is 255. This string of periods is concatenated to the value of the last positional parameter passed to the EXEC.

For example, if the input parameters are:

```
This is a test
```

Then you code the ARG statement like this:

```
ARG P1 P2 P3
```

And the values of the parameters P1, P2, and P3 are:

Table 2. REXX EXEC Parsing Example 1

Positional Parameter	Parameter Value
P1	This
P2	is
P3	a test . . . . . (and so on)

To avoid this string of periods, code a single period (.) or any valid variable name after the last variable name in the template; for example:

```
ARG P1 P2 P3 .
```

This eliminates the string. Then the values of the parameters P1, P2, and P3 are:

Table 3. REXX EXEC Parsing Example 2

Positional Parameter	Parameter Value
P1	This
P2	is
P3	a

If fewer values are to be passed to the EXEC than there are parameters specified, the extra parameters are filled in with a dummy value of . (period). It is not necessary to use each symbolic parameter in the logic section of the EXEC.

In AutoOPERATOR, EXECs can be invoked in seven ways. The information (or input) passed to the REXX EXEC varies depending on how the EXEC is invoked. The input passed to the positional parameters can be different if an EXEC is invoked by a Rule (Rule-initiated EXECs) or by a user (user-initiated EXECs).

Following is an example ARG statement for an EXEC named PAYROLL which starts or stops a payroll application when a user schedules the EXEC:

```
ARG PAYROLL P1
```

To invoke the EXEC, enter its name (PAYROLL) and the parameter value (START or STOP) on the COMMAND line of any AutoOPERATOR panel. AutoOPERATOR searches BBPROC and executes the EXEC when it finds a member named PAYROLL. It passes a START or STOP value to the P1 positional parameter and passes the EXEC name, PAYROLL, as the first positional parameter in the variable named PAYROLL.

AutoOPERATOR does not do the parsing of the message text for Message-initiated EXECs. For example, to parse the message:

```
E JOBNAME,PERFORM=999
```

you must code the REXX EXEC as:

```
/* REXX */
PARSE ARG P1 P2 ',' P3 ',' P4
```

The result is:

P1 = E
P2 = JOBNAME
P3 = PERFORM
P4 = 999

The following table lists where you can find complete discussions of each type of REXX EXEC and the parameters that are passed to them:

To read about...	**See page...**
Rule-initiated EXECs	29
ALERT-initiated EXECs	31
User-initiated EXECs	36
Time-initiated EXECs	38
EXEC-initiated EXECs	42
Externally initiated EXECs	44
End-of-Memory—initiated EXECs	46

Documenting REXX EXECs

As discussed in “Passing Data” on page 24, the ARG statement identifies the parameters that the subsequent IMFEXEC commands and EXEC statements process.

Following the ARG statement, you should have a section that uses comment statements to describe the symbolic parameters. A comment statement looks like:

```
/*  This is an example of a comment in an EXEC  */
```

This comment section is optional but highly recommended because it provides consistency and helps other system administrators, analysts, or operators who use or maintain the EXEC. The comment section explains the purpose of the EXEC and the expected values to be passed to each symbolic parameter defined by the ARG statement.

Figure 1 shows an example of the ARG statement and comment section for a user-initiated REXX EXEC named PAYROLL.

```
/* REXX EXEC */
ARG PAYROLL P1
/*-------------------------------------------------------------------*/
/*         DOC    GROUP (MVS)                                        */
/*         DOC    FUNC (PAYROLL)                                     */
/*         DOC    CODE (PY)                                          */
/*         DOC    DESC (Start/Stop PAYROLL Application)              */
/*         DOC  AUTHOR (JAC)                                         */
/*-------------------------------------------------------------------*/
/* EXEC Description:  This sample EXEC, named PAYROLL, starts or  */
/* stops the payroll application when the EXEC name, PAYROLL,along*/
/* with a START or STOP parameter, is entered in the command input*/
/* line of an AutoOPERATOR panel.                                    */
/*-------------------------------------------------------------------*/
/* Symbolic Parameter Definitions:                                   */
/*                                                                   */
/* EXECNAME  The member name for this EXEC in the SYSPROC            */
/*           concatenated data set. The value for EXECNAME           */
/*           is PAYROLL.                                             */
/*                                                                   */
/* P1      The value for P1 is either START or STOP.                 */
/*-------------------------------------------------------------------*/
```

Figure 1. Sample Comment Section for a REXX EXEC

Writing the Logic Section

The logic section of a REXX EXEC is a combination of programming elements such as:

Element type	**For example:**
TSO REXX assignment statements	ARG, PARSE, PULL
TSO REXX Control statements	CALL, EXIT, ITERATE
TSO REXX Built-in functions	DATE(), SUBSTR(), WORD()
AutoOPERATOR **variables**	QIMFID, QSMFID, QJNLSTA

and AutoOPERATOR IMFEXEC statements that enable you to write automation procedures. The concept is identical to programming in other languages such as COBOL and PL/I, except that REXX EXECs are not compiled prior to execution.

This chapter describes passing parameters to AutoOPERATOR REXX EXECs. For complete information about writing REXX EXECs, refer to the IBM publication *TSO Extensions Version 2: REXX/MVS User's Guide*.

Describing AutoOPERATOR REXX EXECs

The following sections describe the different AutoOPERATOR REXX EXECs based on how they can be invoked in AutoOPERATOR and how information is passed to the ARG statement.

Rule-Initiated REXX EXECs

An EXEC is Rule-initiated if its name is specified in the `EXEC Name/Parms` field of the Rule Processor Action Specification panel of a fired rule.

Refer to the Rule Processor chapters in the *MAINVIEW AutoOPERATOR Basic Automation Guide* for more information about writing Rules and how to write a Rule that schedules an EXEC.

Potential Use

EXECs scheduled by a Rule through the Rule Processor application can perform automation that cannot be performed by a Rule. For example, a Rule-initiated EXEC can, based on the text of a message, issue ALERTs, submit other EXECs, or invoke SYSPROG services. In general, use Rule-initiated EXECs to perform advanced automation as a result of a message.

Parameters Passed to the EXEC

The individual words of the message that caused a Rule to fire are passed as input to the EXEC. A word is any character string separated by a blank or a comma.

Example of input:

The message:

```
$HASP103 CMFTEXT BAB031
```

is an example of a message that can cause a Rule to fire. If the Rule has an EXEC associated with it, then the words of this message are passed as parameters to the ARG statement of the EXEC.

Specifying Additional Parameters

From the Rule Processor Action Specification panel, you also can specify additional parameters you want to send to the EXEC. This is done on the `EXEC/Parms` field of any Action Specification panel.

Note that the first parameter specified in this field becomes the first parameter passed to the EXEC. Subsequent parameters are passed to the EXEC in the order they were entered.

This means the message ID and any message text will not be passed to the EXEC. To have the message ID and any message text passed to the EXEC, the Rule must use the IMFTEXT variable.

Example

The following is an example of a Rule-initiated EXEC scheduled by the Rule handling the $HASP103 message.

```
/*REXX EXEC */
ARG MSGID SETUP W2 W3
/*-------------------------------------------------------------------*/
/* DOC GROUP(MVS) FUNC(JES2) CODE(J2)                                */
/* DOC DISP(YES) AUTHOR(B&B)                                         */
/* DOC DESC(RESPOND TO $HASP103 AND WRITE MESSAGE TO JOURNAL)        */
/*-------------------------------------------------------------------*/

"IMFEXEC MSG 'JOB "SETUP" IS REQUESTING "W2"'"

EXIT
```

Figure 2. Rule-Initiated REXX EXEC Example

The positional parameters passed to the ARG statement of the Rule-initiated EXEC are shown in the following table:

Positional Parameter	Variable Name	Value Passed	Description of Value Passed
1	MSGID	$HASP103	Is the message ID of the message that fired the Rule that calls this EXEC
2	SETUP	CMFTEXT	Is the name of the job requesting a tape mount
3	W2	BAB031	Is the volume serial number of the tape to be mounted
4	W3	.	Is a dummy value used to fill in for the fourth parameter that was not passed with the message

Describing the Example

This EXEC issues the IMFEXEC MSG command to write a message to the BBI-SS PAS Journal that, when all the values from the input are substituted for the ARG statement parameters, translates into:

```
JOB CMFTEXT IS REQUESTING BAB031
```

For information about Rule-initiated EXECs and retrieving information from MVS multi-line WTOs or IMS multi-segment messages, refer to “Rule-Initiated EXECs Initiated by MVS Multi-Line or Multi-Segment Messages” on page 71.

ALERT-Initiated REXX EXECs

An ALERT-initiated EXEC (also called a follow-up EXEC) is scheduled by a user from the ALERT Management Facility. When coding the EXEC that issues the IMFEXEC ALERT command, use the EXEC parameter to specify the name of the follow-up EXEC.

The EXEC is then scheduled from the ALERT Detail panel of the ALERT Management Facility by entering any value (up to three characters) in the `RSP` column of the panel.

To read about ...	**Refer to ...**
How to actually invoke the EXEC	Chapter 3, the “ALERT Management Facility” in the *MAINVIEW AutoOPERATOR Basic Automation Guide*
About coding an ALERT with an associated EXEC	Chapter 6, “Using the IMFEXEC Command Statements” in this book

Potential Use

When an ALERT appears on the DETAIL display, it may require an advanced automation response. An ALERT-initiated EXEC can handle such a response. By entering any value (up to three characters) in the `RSP` column of the ALERT Detail panel, you can schedule a follow-up EXEC.

One possible use for an ALERT-initiated EXEC is to log messages in the BBI-SS PAS Journal.

Parameters Passed to the EXEC

When an ALERT-initiated EXEC is coded, the `IMFEXEC ALERT . . . EXEC(ABC)` command can schedule the follow-up EXEC with or without parameters. In this example, the EXEC name is `ABC`:

- **Without** optional parameters:

  ```
  "IMFEXEC ALERT ... EXEC(ABC)"
  ```

- **With** optional parameters (`x y z`):

  ```
  "IMFEXEC ALERT ... EXEC('ABC x y z')"
  ```

  If the EXEC has parameters, you **must** enclose them in single quote marks (`' '`) with the EXEC name. If you do not, only the EXEC name will be passed and the parameters will not be passed.

  See the two examples of input on page 32 for more information.

The first positional parameter passed to the ALERT-initiated EXEC is always the EXEC name. The characters that you enter in the `RSP` column ALERT Detail Display to schedule the EXEC are also passed. However, the position that those characters have depends on whether or not you use optional parameters.

Example of input without parameters

For example, the user enters:

```
DEF
```

in the RSP column of the ALERT DETAIL DISPLAY panel.

Then, the ARG statement receives data passed in the following way:

Positional parameter	Value passed	Description of value passed
1	EXEC name	Is the name of the EXEC
2	DEF (contents of RSP column)	Is the (up to) three character string the user enters in the RSP column of the ALERT DETAIL DISPLAY panel to actually invoke the ALERT
3 through n	Text of the ALERT	Are the actual words of the ALERT asoociated with the invoked EXEC
n + 1	....	The period pads the positional parameter

Example of input with parameters

For example, the user enters:

```
DEF
```

in the RSP column of the ALERT DETAIL DISPLAY panel.

Then, the ARG statement receives data passed in the following way:

Positional parameter	Value passed	Description of value passed
1	EXEC name	Is the name of the EXEC
2	x	Is the first parameter passed to the EXEC
3	y	Is the second parameter passed to the EXEC
4	z	Is the third parameter passed to the EXEC
5	DEF (contents of RSP column)	Is the (up to) three character string the user enters in the RSP column of the ALERT DETAIL DISPLAY panel to actually invoke the ALERT
6 through n	Text of the ALERT	Are the actual words of the ALERT associated with the invoked EXEC
n + 1	....	The period pads the positional parameter

Example 1: ALERT-Initiated EXEC without Optional Parameters

This example shows an IMFEXEC ALERT statement that schedules an EXEC named SETJOB without any optional parameters:

```
"IMFEXEC ALERT KEYSETUP 'SETUP BAB031 . . . JOB 00395' EXEC(SETJOB)",
         "QUEUE(ABC) PRI(INFO)"
```

The ALERT generated by this statement looks like:

```
RSP TIME  IND ORIGIN  ------------------------------------------
___ 10:15 e   SYSB    SETUP BAB031 . . . JOB 00395
```

The user enters OUT (or any up to three-character string) in the RSP column. The positional parameters passed to the ALERT-initiated EXEC in this example are defined in the following table.

Positional Parameter	Variable Name	Variable Passed	Description of Variable Passed
1	EXECNAME	SETJOB	Is the name of the EXEC
2	RSP	OUT (contents of RSP column)	Is the (up to) three character string the user enters in the RSP column of the ALERT DETAIL DISPLAY panel to actually invoke the ALERT
3	ATEXT1	SETUP	First word of ALERT text
4	ATEXT2	BAB031	Second word
5	ATEXT3	.	Third word
6	ATEXT4	.	Fourth word
7	ATEXT5	.	Fifth word
8	ATEXT6	JOB	Sixth word
9	ATEXT7	00395	Is the last word of the ALERT text
10	.	.	The period pads the positional parameters

```
/* REXX EXEC */
ARG EXECNAME RSP ATEXT1 ATEXT2 ATEXT3 ATEXT4 ATEXT5 ATEXT6 ATEXT7 .
/*-------------------------------------------------------------*/
/* DOC GROUP(MVS) FUNC(JES2) CODE(J2)                          */
/* DOC DISP(YES) AUTHOR(B&B)                                   */
/* DOC DESC(WRITE MESSAGE FOR SETUP)                           */
/*-------------------------------------------------------------*/

"IMFEXEC MSG 'ALERT "EXECNAME" IS REQUESTING SETUP FOR JOB "ATEXT7"'"

EXIT
```

Figure 3. ALERT-Initiated REXX EXEC Example 1

Describing the Example

This EXEC issues the IMFEXEC MSG command to write a message to the BBI-SS PAS Journal that, when all the values from the input are substituted for, translates into:

```
ALERT SETJOB IS REQUESTING SETUP FOR JOB 00395
```

Example 2: ALERT-Initiated EXEC with Optional Parameters

This example shows an IMFEXEC ALERT statement that schedules an EXEC named SETJOB with the optional parameter IMMEDIATE:

```
"IMFEXEC ALERT KEYSETUP 'SETUP BAB031 . . . JOB 00395'",
     "EXEC('SETJOB IMMEDIATE')"
```

The ALERT generated by this statement looks like:

```
RSP TIME  IND ORIGIN  ------------------------------------------
___ 10:15 e   SYSB    SETUP BAB031 . . . JOB 00395
```

The user enters OUT (or any up to three-character string) in the RSP column. The positional parameters passed to the ALERT-initiated EXEC in this example are defined in the following table.

Table 4. Example of ALERT-initiated EXEC Parameters and Variables

Positional Parameter	Variable Name	Variable Value	Description of Variable Value
1	EXECNAME	SETJOB	Is the name of the EXEC
2	TIME	IMMEDIATE	Is the optional parameter passed to the EXEC to specify when the job should be run
3	RSP	OUT (contents of RSP column)	Is the (up to) three-character string the user enters in the RSP column of the ALERT DETAIL DISPLAY panel to actually invoke the ALERT
4	ATEXT1	SETUP	First word of ALERT text
5	ATEXT2	BAB031	Second word of ALERT text
6	.	.	The period pads the positional parameters

```
ARG EXECNAME TIME RSP ATEXT1 ATEXT2 .
/*-----------------------------------------------------------------*/
/* DOC GROUP(MVS) FUNC(JES2) CODE(J2) DOC DISP(YES)                */
/* AUTHOR(B&B) /* DOC DESC(WRITE MESSAGE FOR SETUP & TIME)         */
/*-----------------------------------------------------------------*/

"IMFEXEC MSG
   'ALERT "EXECNAME" IS REQUESTING SETUP FOR "ATEXT2" AT "TIME"'"
EXIT
```

Figure 4. ALERT-Initiated REXX EXEC Example 2

Describing the Example

This EXEC issues the IMFEXEC MSG command to write a message to the BBI-SS PAS Journal that, when all the values from the input are substituted for, translates into:

```
ALERT SETJOB IS REQUESTING SETUP AT IMMEDIATE FOR JOB 00395
```

User-Initiated REXX EXECs

A user-initiated EXEC (also known as a command-initiated EXEC) is scheduled when a user enters the EXEC name from the BBI terminal session (TS) command line with the command prefix of % or 4.

You also can schedule a user-initiated EXEC by issuing a MVS MODIFY command against a BBI PAS subsystem (BBI-SS PAS); for example:

```
F SYSB,%EXECB
```

Finally, you also can use the AutoOPERATOR EXEC Manager application to issue a user-initiated EXEC. Refer to the *MAINVIEW AutoOPERATOR Basic Automation Guide* for more information.

Potential Use

Use user-initiated EXECs when you want to schedule an EXEC from a TS or an MVS console. The example in this section shows how to schedule an EXEC named START for execution. This EXEC is used to vary a VTAM node online.

Parameters Passed to the EXEC

The first positional parameter is the 1- to 8-character EXEC name (in this case, `START`). Any of the positional parameters are optional.

Example of input:

To use the EXEC named START, enter the following command on any TS command line:

```
%START termid
```

where `termid` is the name of the VTAM node you specify to bring online. For example, this termid value could be BS4000. The command would look like:

```
%START BS4000
```

Example

The following shows an example of an EXEC that would be scheduled:

```
/* REXX EXEC */
ARG START TERMID .
/*-------------------------------------------------------------------*/
/* DOC GROUP(MVS) FUNC(VTAM) CODE(VT)                                */
/* DOC DISP(YES) AUTHOR(B&B)                                         */
/* DOC DESC(ACTIVATE THE NODE)                                       */
/*-------------------------------------------------------------------*/

"IMFEXEC CMD #VARY NET,ACT,ID="TERMID""

EXIT
```

Figure 5. User-Initiated REXX EXEC Example

The positional parameters passed to the ARG statement of the user-initiated EXEC are shown in the following table:

Positional Parameter	Variable Name	Value Passed	Description of Value Passed
1	START	START	Is the EXEC name
2	TERMID	BS4000	Is the name of the terminal
3	.	.	The period pads the positional parameters

Describing the Example

In this example, the IMFEXEC CMD statement is used to issue a VTAM command to vary a terminal online. Refer to “CMD (Issue IMS Command without Response)” on page 315 and “CMD (MVS Version with Response through X-MCS Consoles)” on page 310 for more information about the IMFEXEC CMD command and MVS commands.

Time-Initiated REXX EXECs

Time-initiated EXECs are invoked when:

- An EXEC name is specified in the AutoOPERATOR TIMEXEC application.

  These EXECs are invoked by AutoOPERATOR Timer Facility when the user-defined time condition occurs. Refer to the section called “TIMEXEC Application” in the *MAINVIEW AutoOPERATOR Basic Automation Guide*.

- A BLK request is issued
- An EXEC-initiated EXEC uses the CALLX service

  For example, by coding:

  ```
  "IMFEXEC IMFC SET REQ=CALLX @HOURLY START=06:00:00 STOP=16:00:00
     I=01:00:00"

  EXEC @HOURLY will execute every hour, beginning at 6:00 am and
  ending at 4:00 pm.
  ```

- The @TIMER sample solution is used (refer to the *MAINVIEW AutoOPERATOR Basic Automation Guide* for more information).

Potential Use

Any production environment that follows a daily schedule requires specific jobs to start and stop at the same time every day. Using the AutoOPERATOR Timer Facility, you can have EXECs automatically scheduled at specific times to perform automation tasks or react to certain activities.

Parameters Passed to the EXEC

Time-initiated EXECs have specific information passed to the 11 positional parameters as described in this table.

Table 5. Time-Initiated EXEC Parameters and Values

Positional Parameter	Description of Parameter Value
1	EXECNAME - 1 to 8 character name of this EXEC.
2	1- to 8-character target name.
3	**AutoOPERATOR for IMS only.** This is the 4-character IMS ID used by AutoOPERATOR for IMS only. This variable must be coded; however, its value is unpredictable for AutoOPERATOR for CICS and AutoOPERATOR for MVS.
4	4-character BBI-SS PAS Subsystem identifier.
5	Current Gregorian date in mm/dd/yy format.

Table 5. Time-Initiated EXEC Parameters and Values (Continued)

Positional Parameter	Description of Parameter Value
6	The time the EXEC is scheduled. The time is in the hours:minutes:seconds format of hh:mm:ss. This is the time when the timer-driven request interval expires. In a congested system, the actual EXEC execution could be delayed because of MVS dispatching priorities.
7	Day of the week is a digit, where 1 is Monday, 2 is Tuesday, 3 is Wednesday, 4 is Thursday, 5 is Friday, 6 is Saturday, and 7 is Sunday.
8	Current Julian date in yyddd format.
9	**MAINVIEW AutoOPERATOR for IMS only.** Not used by MAINVIEW AutoOPERATOR for CICS or MVS. This variable must be coded; however, its value is unpredictable for MAINVIEW AutoOPERATOR for CICS and MAINVIEW AutoOPERATOR for MVS. This is the elapsed time that IMS/VS has been active in the total hours:minutes:seconds format of hhh:mm:ss. This is the elapsed control region job time, not the elapsed time since the first IMS/VS checkpoint. If IMS/VS is not active, the value is 000:00:00.
10	**MAINVIEW AutoOPERATOR for IMS only.** Not used by MAINVIEW AutoOPERATOR for CICS or MVS. This variable must be coded; however, its value is unpredictable for MAINVIEW AutoOPERATOR for CICS and MAINVIEW AutoOPERATOR for MVS. The IMS/VS restart type, as follows: **ERE** Emergency restart **WARM** Warm restart **COLD** Cold restart **INACT** IMS/VS is not active. This value is also passed during: – IMS/VS initialization until the first checkpoint is taken – IMS/VS termination after the shutdown checkpoint is issued It remains INACT until IMS/VS restarts and the first checkpoint is taken.
11	**MAINVIEW AutoOPERATOR for IMS only.** Not used by MAINVIEW AutoOPERATOR for CICS or MVS. This variable must be coded; however, its value is unpredictable for MAINVIEW AutoOPERATOR for CICS and MAINVIEW AutoOPERATOR for MVS. A 1- to 5-digit number for the numer of times the EXEC has been invoked. The P10 value is reset to 1 every time the P9 status changes.

It is not always necessary to identify all 11 parameters on the PROC statement. For example, an EXEC may only require positional parameter eight (the current Julian date). In this case only the first eight parameters need to be coded on the PROC statement. The required PROC would be:

```
ARG SUBJOB P1 P2 P3 P4 P5 P6 P7 P8
```

Example

```
/* REXX EXEC */
ARG EXECNAME
 /*-------------------------------------------------------------*/
 /*EXEC Description:  This sample EXEC displays the status of your*/
 /*                   system.                                   */
 /*-------------------------------------------------------------*/
 /*Positional Parameter Count:                                  */
 /*                                                            */
 /*11        The total number of ARG  parameters.  This value will*/
 /*          always be 11 for a time-initiated EXEC.           */
 /*                                                            */
 /*Symbolic Parameter Definitions:                              */
 /*                                                            */
 /*SSTATUS   The BBPROC member name for this EXEC.             */
 /*                                                            */
 /*-------------------------------------------------------------*/

"IMFEXEC CMD .D V,ALL" /* Displays all shared variables        */
"IMFEXEC CMD .D L,ALL" /* Displays of all BBI-SS PAS/BBI-SS PAS    */
Links                                                          */
"IMFEXEC CMD .D R"     /* Displays remote users                */
"IMFEXEC CMD .D A"     /* Displays ACTIVE STATUS               */

 EXIT
```

Figure 6. Time-Initiated REXX EXEC Example

The positional parameters passed to the ARG statement of the time-initiated EXEC are shown in the following table:

Positional Parameter	Variable Name	Value Passed	Description of Value Passed
1	EXECNAME	SSTATUS	Is the name of the EXEC invoked by the timer facility

Describing the Example

This EXEC uses the IMFEXEC CMD command to issue various BBI control commands to be logged to the BBI -SS PAS Journal. The ARG statement is written as the first REXX statement of the EXEC named SSTATUS by specifying:

```
ARG EXECNAME
```

where:

- ARG instructs REXX to process the arguments passed to this REXX EXEC
- EXECNAME is a variable which contains the name of the EXEC

  There is only one positional parameter in this statement, the variable containing the EXEC name. The remaining 10 positional parameters are ignored.

This time-initiated EXEC is scheduled to take a snapshot of the BBI environment. The EXEC uses only one input variable for this task and it issues four BBI control commands so the output is recorded in the BBI-SS PAS Journal. This allows you to review the data.

EXEC-Initiated REXX EXECs

An EXEC-initiated EXEC is scheduled when the IMFEXEC SELECT command is coded, specifying the EXEC parameter. The EXEC parameter names the EXEC to be scheduled along with any parameters; for example:

```
"IMFEXEC SELECT . . . EXEC(execname)"
```

where `execname` is the name of any EXEC to be scheduled.

Potential Use

Use an EXEC-initiated EXEC when you want to:

- Invoke a common EXEC that might be used by several other EXECs
- Schedule another EXEC and have it execute asynchronously

  EXEC-initiated EXECs can be scheduled to execute either synchronously or asynchronously by the calling EXEC. For more information, see“Invoking EXECs Synchronously with IMFEXEC SELECT(EXEC) WAIT(YES)” on page 78 .

Parameters Passed to the EXEC

The first positional parameter is the 1- to 8-character name of the EXEC. Any following positional parameter are optional.

Example of input:

The command:

```
"IMFEXEC SELECT EXEC(START BS4000)"
```

schedules the EXEC called START for execution. An optional parameter containing the value `BS4000` is passed to START as input.

Example

This example shows the calling EXEC that schedules the called EXEC named START:

```
/* REXX EXEC */
/*-------------------------------------------------------------*/
/* DOC GROUP(MVS) FUNC(VTAM) CODE(VT)                          */
/* DOC DISP(YES) AUTHOR(B&B)                                   */
/* DOC DESC(CALL ACTIVATE EXEC)                                */
/*-------------------------------------------------------------*/

"IMFEXEC SELECT EXEC(START BS4000)"

EXIT
```

This example shows the called EXEC:

```
/* REXX EXEC */
ARG START TERMID .
/*-------------------------------------------------------------------*/
/* DOC GROUP(MVS) FUNC(VTAM) CODE(VT)                                */
/* DOC DISP(YES) AUTHOR(B&B)                                         */
/* DOC DESC(ACTIVATE THE NODE)                                       */
/*-------------------------------------------------------------------*/

"IMFEXEC CMD #VARY NET,ACT,ID="TERMID""

EXIT
```

Figure 7. EXEC-Initiated REXX EXEC Example

The positional parameters passed to the EXEC-initiated EXEC are shown in the following table:

Positional Parameter	Variable Name	Value Passed	Description of Value Passed
1	START	START	Is the name of the EXEC
2	TERMID	BS4000	Is the name of the terminal to be started online
3	.	.	The period pads the positional parameters

Describing the Example

The called EXEC in this example receives a parameter from the calling EXEC (BS4000) and uses that value to vary a VTAM node active with the IMFEXEC CMD command. Refer to “CMD (MVS Version with Response through X-MCS Consoles)” on page 310 for more information about the IMFEXEC CMD statement and MVS commands.

Externally Initiated REXX EXECs

Externally initiated EXECs are scheduled by:

- A job step that executes the IMFSUBEX program
- A user-written program
- A TSO user The EXEC that IMFSUBEX schedules is called an externally initiated EXEC.

Potential Use

There are many instances where full automation requires the completion of a task that is not an EXEC and is running outside of the BBI-SS PAS. A database backup is one example. When the backup completes, you can use an externally initiated EXEC to notify AutoOPERATOR to schedule any further actions.

Two possible ways to do this are through writing a Rule and through IMFSUBEX. If you use the Rule Processor application to write Rules, then:

1. Create a message with a unique message-ID
2. Send the message to the operator's console
3. Create a Rule to process the message

If you use the IMFSUBEX facility, you can directly schedule an EXEC to take subsequent automation actions. For more information for how to invoke externally initiated EXECs, refer to “Invoking REXX EXECs from Outside of AutoOPERATOR with IMFSUBEX” on page 93.

Parameters Passed to the EXEC

The first positional parameter is the 1- to 8-character name of the EXEC. Any following positional parameter are optional.

Example of input:

The following JCL shows how the subroutine IMFSUBEX schedules an EXEC named BACKDONE for execution.

```
//STEPX   EXEC  PGM=IMFSUBEX,
//              PARM='SS(SSA1) EXEC(BACKDONE SYST1)'
```

Example

The following EXEC is scheduled:

```
/* REXX EXEC */
ARG BACKDONE V1 .
/*-------------------------------------------------------------------*/
/* DOC GROUP(MVS) FUNC(BKUP) CODE(BK)                                */
/* DOC DISP(YES) AUTHOR(B&B)                                         */
/* DOC DESC(SEND NOTIFY/LOG FOR A SUCCESSFUL BACKUP)                 */
/*-------------------------------------------------------------------*/

"IMFEXEC CMD #SE 'VOLUME "V1" SUCCESSFULLY
DUMPED',LOGON,USER=(SYSP1,SYSP2)"
"IMFEXEC  MSG 'VOLUME "V1" SUCCESSFULLY DUMPED'"

EXIT
```

Figure 8. Externally Initiated REXX EXEC Example

The positional parameters passed to the EXEC-initiated EXEC are shown in the following table:

Positional Parameter	Variable Name	Value Passed	Description of Value Passed
1	BACKDONE	BACKDONE	Is the name of the EXEC invoked
2	V1	SYST1	Is the name of the volume serial number of a DASD
3	.	.	The period pads the positional parameters

Describing the Example

The EXEC named BACKDONE is scheduled in a target subsystem called SSA1. A single parameter is passed (SYST1) which is a DASD volume serial number. The BACKDONE EXEC receives a volume serial number of a DASD from the second positional parameter to IMFSUBEX.

The BACKDONE EXEC first sends a message to two TSO users, SYSP1 and SYSP2, informing them that the volume backup has been successful and then places a message in the BBI-SS PAS Journal recording a successful operation.

End-of-Memory–Initiated REXX EXEC

Use the End-of-Memory EXEC to ensure that critical address spaces do not terminate unnoticed.

Potential Use

Normally, address space termination can be monitored using standard MVS and JES messages. However, there are situations when monitoring based on these messages is not sufficient because an address space may terminate without producing the expected messages. For example, the expected termination messages may not be produced if the MVS FORCE or SYSPROG EXIT command is used or when an initiator abends.

The End-of-Memory EXEC allows AutoOPERATOR to monitor address space termination regardless of how the address space is terminated. This EXEC is scheduled for the following things when the associated events occur:

Batch jobs	Only when the initiator terminates
TSO users	When any TSO user is terminated
Started tasks	When any started task is terminated

There is only one End-of-Memory EXEC for each AutoOPERATOR subsystem. Each time one of the above mentioned events occurs, AutoOPERATOR automatically schedules an EXEC named IMFEOM if it exists in the SYSPROC concatenation.

Parameters Passed to the EXEC

Two parameters are passed to the End-of-Memory EXEC.

- The first parameter contains the fixed string of *EOM*
- The second parameter contains a character string which can have one of two values:

Parameter value	Description
NORMAL	Indicates normal address space termination
ABNORMAL	Indicates address space was terminated by passing it to RTM This may happen when using the SYSPROG EXIT command or the MVS FORCE command. This is not an indication that the address space abended with a system or user abend code.

Example

This first example shows an EXEC called STRT that is invoked by a Rule (a Rule-initiated EXEC). The Rule is fired when the JES2 message $HASP373 is issued for jobname PRODSTC: $HASP373 indicates that the job has started.

```
/*REXX EXEC */
/*--------------------------------------------------------------*/
/* THIS EXEC IS DRIVEN FROM JES2 MESSAGE, $HASP373, FOR STC     */
/* PRODSTC ONLY                                                 */
/*                                                              */
/* EXEC DESCRIPTION: SET VARIABLE "PRODSTKN" TO STOKEN OF PRODSTC */
/*--------------------------------------------------------------*/
PRODSTKN = IMFSTOKN
"IMFEXEC VPUT PRODSTKN"
```

The second EXEC, IMFEOM, is automatically scheduled when any started task or TSO address space terminates or when a batch initiator abends.Describing the Example

```
/* REXX */
ARG IMFEOM STATUS .
/*--------------------------------------------------------------*/
/* THIS EXEC IS DRIVEN FROM END OF MEMORY EXIT                  */
/*                                                              */
/* EXEC DESCRIPTION: DETERMINE IF ADDRESS SPACE TERMINATING IS  */
/*                   "PRODSTC".  IF SO, INFORM THE OPERATOR.    */
/*--------------------------------------------------------------*/

"IMFEXEC VGET PRODSTKN"
IF IMFSTOKN = PRODSTKN THEN DO
   PRODSTKN =''
   "IMFEXEC VPUT PRODSTKN"
   IF STATUS = ABNORMAL THEN   ,
      "IMFEXEC WTO 'PRODSTC ENDED ABNORMALLY'"
END
```

Figure 9. End-of-Memory—Initiated EXECs Example

When the STRT EXEC is scheduled, the local variable IMFSTOKN contains an identifier that uniquely identifies the PRODSTC started task. Since this variable only exists for the life of the EXEC, STRT saves the IMFSTOKN value in the shared variable pool so that it can be used subsequently by the IMFEOM EXEC.

Important
If this procedure will be used for more than one address space, you should use a variable name other than IMFSTOKN in the shared variable pool or else the value IMFSTOKN might be overridden by the other procedures.

When the IMFEOM EXEC is scheduled, IMFSTOKN refers to the address space that is being terminated. The IMFEOM EXEC compares IMFSTOKN to the PRODSTKN value saved previously by the EXEC named STRT. If the values do not match, IMFEOM exits because the address space that is terminating is not one that is being monitored. If the values do match and the parameter passed to IMFEOM indicates abnormal termination, then a WTO (write-to-operator) is issued to notify the operator.

Refer to “TSO Variables Supplied by AutoOPERATOR” on page 54 for more information about AutoOPERATOR-supplied variables.

Chapter 4. Using Variables in REXX EXECs

This chapter discusses:

- Variables and variable pools available to AutoOPERATOR REXX EXECs
- Manipulating information between the variable pools

Overview

Complex EXECs must be able to do much more than issue commands and return control to their callers. An EXEC must be able to request information from AutoOPERATOR (and other products), compare the information, compare the time elapsed since the last observation, and effect changes that other EXECs or products carry out.

This type of logic requires the ability to save information, either temporarily or permanently, in a simple manner so that it can be accessed later by the same EXEC or other EXECs.

To retain this information for EXECs, AutoOPERATOR provides four kinds of variables and variable pools.

Variable Pool Name	Description
TSO variables	Exist for the life of the EXEC. This chapter lists: • AutoOPERATOR–supplied TSO variables • Modifiable TSO variables • Non-modifiable TSO variables
LOCAL variables	LOCAL variables are stored in a pool that can be accessed only by the current EXEC and other EXECs (using IMFEXEC SELECT WAIT(YES)). AutoOPERATOR passes information to an EXEC in this pool. It is also used by AOAnywhere when sharing variables with an invoking EXEC. The LOCAL variable pool is freed when the EXEC ends and its contents are lost.

<table>
<tr><th>Variable Pool Name</th><th colspan="2">Description</th></tr>
<tr><td rowspan="2">Two types of GLOBAL variable pools: SHARED and PROFILE</td><td colspan="2">Can be saved for later executions of the same EXEC or other EXECs.
The use of the expression “GLOBAL variables” in this book refers to both SHARED and PROFILE variables.</td></tr>
<tr><td>SHARED variables

SHARED variables are stored in a pool that is accessible to all EXECs in the BBI-SS PAS . They can be read, modified, created and deleted by any number of EXECs or Rules. Since EXECs can access them simultaneously, their access should be serialized (see IMFEXEC VENQ and VDEQ). These variables exist in storage beyond the life of the EXEC that created them.
AutoOPERATOR creates a number of SHARED variables that contain system-specific information. SHARED variables are accessible to the Rules Processor and remain in memory when the subsystem is terminated. However, they are lost across IPLs or when a subsystem is restarted with the VPOOL=RESET option.
This chapter lists the AutoOPERATOR-supplied variables.</td><td>PROFILE variables

PROFILE variables are similar to SHARED variables with the exception that they are persistent across IPLs and their contents are never lost unless explicitly deleted.

PROFILE variables are not accessable from Rules.</td></tr>
<tr><td colspan="3">Note: Variable names must be at least 1 and not more than 32 characters in length. The contents of any variable cannot exceed 256 characters.</td></tr>
</table>

AutoOPERATOR also provides four IMFEXEC commands for defining, saving, deleting, and retrieving variables using the different variable pools:

VDCL Defines map lists for variables

VPUT Save variables to a pool

VPUTL Saves long variables (up to 32k and 30 characters long) to a pool

VGET Retrieve variables from a pool

VGETL Retrieves long variables (up to 32k and 30 characters long) to a pool

VDEL Remove variables from pools

VDELL Removes long variables (up to 32k and 30 characters long) to a pool

Although three of these commands are similar to three ISPF Dialog commands, they are not identical. Refer to “Using the IMFEXEC Statements” on page 237 for coding details and carefully review the differences before using them.

LOCAL, SHARED and PROFILE variables (in this order) impose a cost to processing overhead. This means that the system uses more resources to preserve the contents of a PROFILE variable than for a LOCAL variable.

LOCAL, SHARED and PROFILE pool variables each come in two flavors: long and short. Short variables are limited to 255 characters in length and their names to 32 characters long. You cannot manipulate a variable with longer content using the IMFEXEC VGET/VPUT/VDEL statements.

Long variables can be up to 32K in length and have a variable name length up to 30 characters. These variables are manipulated using the VGETL/VPUTL/VDELL IMFEXEC statements. Long and short variables are completely independent from each other. A variable that has been set with the VPUT statement cannot be read with the VGETL statements.

Long variables impose greater processing overhead than short variables. If your code, for example, has to remember only the names of persons, you should always choose a short variable. If, however, a variable can foreseeably grow in length beyond the 255 character limit (say, you might want to concatenation hundreds of volsers into one variable) then you should use the long variable format.

In addition:

- REXX EXECs cannot use any variables that have not been explicitly retrieved into the function pool using IMFEXEC VGET(L) statements.
- A variable with the same name but of different type (long or short) or in different pools (LOCAL/SHARED/PROFILE) can contain completely separate values.
- A LONG variable (set with the VPUTL statement) cannot be retrieved with a short variable operation (VGET) even if the contents of the explicit LONG variable does not exceed the 255 character limit.

For example:

```
Fred='This is a test'
"IMFEXEC VPUT FRED"
"IMFEXEC VGETL FRED"
```

These statements yield:

```
Fred='My name is Fred'
"IMFEXEC VPUT FRED"

Fred="My name is Flintstone"
"IMFEXEC VPUTL FRED"
```

In this case, the variable `Fred` exists both as a long and a short variable, with different contents.

The following table lists where you can find more information about variable pools in this chapter.

Using a TSO Variable Pool

A TSO variable pool is created when an EXEC starts and is deleted when the EXEC ends. The variables in the TSO variable pool can be created in one of two ways:

- Assigning a variable:

  For example, the statement:

  ```
  I=1
  ```

  creates a TSO variable I with a value of 1 in the pool.

- Using IMFEXEC VGET commands

  To access a variable from the LOCAL, SHARED, or PROFILE pools, you must use the IMFEXEC VGET command in the EXEC and move the variable into the TSO pool. The REXX EXEC can perform operations on the value of the variable only when it is in the TSO variable pool. Refer to "VGET" on page 390 for information on coding an IMFEXEC VGET statement.

TSO variables also exist as:

- AutoOPERATOR-supplied variables
- TSO-supplied modifiable and non-modifiable control variables
- Variables that are substituted into the positional parameters on a REXX ARG statement

  Refer to "Passing Data" on page 24 for more information about the ARG statement.

The following sections list the AutoOPERATOR-supplied variables and the modifiable and non-modifiable variables supplied by TSO.

TSO Variables Supplied by AutoOPERATOR

The following lists the TSO variables provided by AutoOPERATOR:

Variable Name	Description	Applicable specifically for which EXEC type
IMFACCTG	Contains all accounting fields for a particular event. The accounting field values are separated by blanks. Maximum length is 142.	Rule-initiated EXECs only
IMFALID	The alarm ID associated with an alarm created by MainView Alarm Manager.	Rule-initiated EXECs only
IMFALPRI	The user-assigned priority of the alarm. Possible values are: **1** Critical **2** Major **3** Minor **4** Warning **5** Informational **6** Clearing	Rule-initiated EXECs only
IMFALQID	The name of the queue to which the alarm was assigned.	Rule-initiated EXECs only
IMFALRM	Contains either Y (sound an alarm) or N (do not sound an alarm).	Rule-initiated EXECs only
IMFCC	The condition code set for each IMFEXEC statement. IMFCC = 00 Normal completion. IMFCC = 04 Warning condition, not necessarily an error. IMFCC = 08 Exception condition or command not found. IMFCC = 12 Error condition. Did not complete operation. IMFCC = 16 Error condition. IMFCC = 20 Severe error condition. Refer to the specific IMFEXEC statement for the exact codes.	All EXEC types
IMFCNTXT	The name of the context of the alarm.	Rule-initiated EXECs only
IMFCONID	Console ID of the message, if message was issued for a specific console. Valid only for messages captured through the Rule Processor application.	Rule-initiated EXECs only

Variable Name	Description	Applicable specifically for which EXEC type
IMFCONNM	Console name to which the WTO was issued. Valid only for MVS SP4 and above. IMFCONNM may be used to identify the origin of an MVS command. The contents of the variable (by origin) are: **Origin** **IMFCONNM Value** Rule Internal SDSF TSO user ID that issued the command Console Console Name where the command was issued	Rule-initiated EXECs only
IMFDAY	Three-character day of the week: MON, TUE, WED, THU, FRI, SAT, SUN.	All EXEC types
IMFDDNAM	Contains the DDNAME specified by the user to generate an external events (EXT event type). EXT events are generated by using the SUBSYS= parameter on a DD statement in JCL. Refer to "EXT Events" in the *MAINVIEW AutoOPERATOR Basic Automation Guide* for more information about EXT events.	Rule-initiated EXECs only
IMFDOMID	The DOM ID associated with a WTO that initiated an EXEC.	Rule-initiated EXECs only
IMFEID	The EXEC identification number, 1 to 99999, assigned to each execution by the EXEC manager. The EXEC MANAGER will not assign the same number to two EXECs in the running or deferred queues, except an EXEC selected with WAIT=(YES) has the same IMFEID as the calling EXEC.	All EXEC types
IMFENAME	Name of EXEC.	All EXEC types
IMFEROUT	A list of routing codes that were assigned to the WTO that triggered the EXEC, such as `1 2 5 9`. This variable is defined only for EXECs initiated as a result of a WTO. IMFEROUT supports return codes up to 128.	Rule-initiated EXECs only

Variable Name	Description	Applicable specifically for which EXEC type
IMFETYPE	The event type that caused the Rule to fire. If a Rule invokes an EXEC, IMFETYPE contains the value from the Rule that invoked the EXEC. Possible values for IMFETYPE are as follows: • MSG • CICS • CMD • JRNL • IMS • ALRT • DB2 • TIME • ALRM • EXT • VAR • MQS • JES3 For more information about these event types, refer to “Describing Events” in the *MAINVIEW AutoOPERATOR Basic Automation Guide*.	Rule-initiated EXECs only
IMFEVFRD	The number of Rules that have fired for a specific event.	Rule-initiated EXECs only
IMFGROUP	The RACF group ID for the address space that issued the message. The group ID is taken from the GROUP= parameter on the job card.	Rule-initiated EXECs only
IMFJCLAS	Job class name from the job card of the batch job that has generated the message.	Rule-initiated EXECs only
IMFJNUM	The JES job number of the job, STC, or TSU that issued the message. It is a fixed length five-digit or a variable length value depending on the setting of the IMFJNUM option in member AAOPRMxx. IMFJNUM can also contain blanks (one or five characters as appropriate) for WTOs that are issued by non-JES tasks, such as a STC started under MSTR. When IMFJNUM=5 (the default setting) and the job number is greater than 99,999 (for example, T0100000, S0999999, etc.) are encountered, IMFJNUM will be null (zero length).	Rule-initiated EXECs only
IMFJTYPE	Type of job issuing message: **J** Batch Job **T** TSO User **S** Started Task	Rule-initiated EXECs only
IMFLPROD	The name of the product associated with the alarm.	Rule-initiated EXECs only

Variable Name	Description	Applicable specifically for which EXEC type
IMFLTYPE	A literal value associated with the alarm; possible values can be START or STOP.	Rule-initiated EXECs only
IMFLUSER	The user-specified user ID associated with the alarm.	Rule-initiated EXECs only
IMFMPFAU	Contains the value of a message from the MPF AUTO keyword. Use this variable to determine the value of the MPF AUTO keyword for a message.	Rule-initiated EXECs only
IMFMPFSP	Contains the value of a message from the MPF SUP keyword. Use this variable to determine the value of the MPF SUP keyword for a message.	Rule-initiated EXECs only
IMFMSTYP	Contains a 2-character variable for the message type. This variable is only for the CMD and MSG event types. Valid values for the first character are: **N** A regular WTO **W** A regular WTOR **M** A major line of a multi-line WTO (MLWTO) Valid values for the second character are: **C** Command **R** Command response	Rule-initiated EXECs only
IMFNOL	Number of lines in WTOR that caused the EXEC to be invoked or the number of lines returned from a service. This value is limited to 9999 lines for Rule-initiated EXECs and for data returned by IMFEXEC CMD with response.	All EXEC types
IMFOASID	Originating Address Space ID (ASID) of the message. For IMFEOM, it is set to the ASID that is being terminated. For ORIGIN=JRNL, it is set to the subsystem ASID name.	Rule-initiated EXECs only
IMFODATE	Date when the message or alarm was issued. Valid only for messages captured through the Rule Processor. The date format is in Julian calendar format; for example: 95.100, where: **95** Are the last two digits of the year 1995. **100** Is the 100th day of the year. In a non-leap-year, this is equal to March 10.	Rule-initiated EXECs only
IMFODESC	A list of descriptor codes assigned to the WTO that triggered the EXEC, such as 2 11. This variable is defined only for EXECs initiated as a result of a WTO.	Rule-initiated EXECs only

Variable Name	Description	Applicable specifically for which EXEC type
IMFOJOB	For WTOs, IMFOJOB contains the job or started task that issued the WTO. For CICS messages, IMFOJOB contains the CICS region name that the subsystem issued the message for, which is useful when monitoring multiple CICS regions with one BBI-SS PAS . For DB2 messages, IMFOJOB contains the DB2 region name that the subsystem issued the message for, which is useful when monitoring multiple DB2 regions with one BBI-SS PAS. For IMS messages, IMFOJOB contains: • The IMS job name for IMS MTO messages • The IMS job name for commands (and their responses) entered from AutoOPERATOR • The originating LTERM for commands (and their responses) entered from an IMS LTERM For BBI-SS PAS Journal messages issued by an EXEC, IMFOJOB contains the user ID of the person who invoked the EXEC. For Journal messages issued by MainView for DB2, IMFOJOB contains the name of the DB2 Region for which the message was issued. For Time-initiated EXECs, IMFOJOB contains the user ID associated with that EXEC. This may be the user ID passed on the command or it may default to the value of the AUTOID keyword specified in BBPARM member BBIISP00.	All EXEC types
&IMFXOJOB	Contains the name of the original job or started task that requested the WTO to be issued by another address space. The contents of IMFXOJOB are only meaningful if the WTO is issued by another address space, otherwise its contents are identical to IMFOJOB.	All EXEC types initiated by a MSG type rule
IMFOQID	CICS transient data queue name if source of message is CICSTD.	Rule-initiated EXECs only

Variable Name	Description	Applicable specifically for which EXEC type
IMFORGN	Origin of EXEC-Job name/USERID causing EXEC to be invoked. For EXECs triggered by the Rule Processor, IMFORGN contains the BBI-SS PAS ID. This is so that EXECs invoked on remote systems that are triggered by message filters on the local system can use authorized services, such as SYSPROG services. Security checking is done against a BBPARM member in the remote system with the name of the BBI-SS PAS ID. See “Determining the Origin of a Command or EXEC” on page 91 for a discussion about using IMFORGN to determine the origin of an EXEC across BBI-SS PASs.	All EXEC types
IMFORGSS	The BBI subsystem ID of the BBI-SS PAS that originated an EXEC. If originated locally, IMFORGSS is the same as QIMFID. See “Determining the Origin of a Command or EXEC” on page 91 for a discussion about using IMFORGN to determine the origin of an EXEC across BBI-SS PASs.	All EXEC types
IMFOROUT	A list of routing codes that were assigned to the WTO that triggered the EXEC, such as `1 2 5 9`. This variable is defined only for EXECs initiated as a result of a WTO. IMFOROUT supports return codes up to 16.	Rule-initiated EXECs only
IMFOTIME	Time when the message was issued. Valid only for messages (also known as events) captured through the Rule Processor. The valid form of the variable is hh:mm:ss for all Rule event types except for the MSG event type. For MSG events, the valid form of the variable is hh.mm.ss. For the ALRM events, the time represents the time the exception occurred.	Rule-initiated EXECs only
IMFPCMD	The PCMD associated with the alarm.	Rule-initiated EXECs only
IMFPOST	A 1 to 255 character code received from an EXEC that issues the IMFEXEC POST command against an ECB with the same name that the current EXEC is waiting on.	All EXECs
IMFPRIO	Contains the dispatching priority of the currently running EXEC after the IMFEXEC CHAP command has been issued.	All EXECs
IMFRC	The return code set by a called EXEC with WAIT(YES) or the return code set by a non-AutoOPERATOR command or program. Refer to “Understanding Completion Codes for EXEC-Initiated EXECs with WAIT(YES) and User Written Programs” on page 354 for a more detailed discussion.	EXEC-initiated EXECs only

Variable Name	Description	Applicable specifically for which EXEC type
IMFREPLY	Reply ID of the WTOR message. Valid only for messages captured through the Rule Processor.	Rule-initiated EXECs only
IMFRLFRD	The number of times a Rule was fired.	Rule-initiated EXECs only
IMFRLID	The Rule identifier that fired an EXEC.	Rule-initiated EXECs only
IMFRLMAT	The number of times the Rules search criteria was matched.	Rule-initiated EXECs only
IMFRLSET	The name of the Rule Set the Rule belongs to.	Rule-initiated EXECs only
IMFRLSTA	The Rule status: **TEST** Indicates that the status of the Rule that invoked the EXEC is in an TEST state. **ACTIVE** Indicates that the status of the Rule that invoked the EXEC is in ACTIVE state.	Rule-initiated EXECs only
IMFRUSER	The RACF user ID for the address space that issued the message. The user ID is taken from the USER= parameter on the job card.	Rule-initiated EXECs only
IMFSCOPE	The name of the scope associated with the alarm.	Rule-initiated EXECs only
IMFSTOKN	The Address Space STOKEN. This name is unique for the life of the IPL.	Rule-initiated EXECs or End-of-Memory initiated EXECs
IMFSYSID	Originating job name For CICS messages, IMFSYSID contains the BBI started task name.	Rule-initiated EXECs only
IMFTEXT	The character text that caused the EXEC to be scheduled.	All EXEC types
IMFTOKEN	Token ID of the message. Same as hardcopy ID. Used to attach MLWTO Minor/Major Lines. Valid only for messages captured through the Rule Processor.	Rule-initiated EXECs only
IMFVIEW	The name of the view associated with the alarm.	Rule-initiated EXECs only
IMFWTDOM	The DOM ID associated with a WTO issued by IMFEXEC WTO command.	All EXEC types

Variable Name	Description	Applicable specifically for which EXEC type
IMFWTCON	Created when a reply is successfully received. The eight-character name of the console from which the reply to the WTOR was entered. One of the possible uses for the IMFWTCON variable is that it enables you to direct reply WTOs specifically to the console where the user entered the reply to this WTOR.	
Note: These variables are carried over to the TSO pool created for an EXEC called using the IMFEXEC SELECT command with parameter WAIT(YES) specified. See “Invoking EXECs Synchronously with IMFEXEC SELECT(EXEC) WAIT(YES)” on page 78 for more information about EXEC-initiated EXECs executing within the same thread.		

TSO Modifiable Control Variables

AutoOPERATOR REXX EXECs support the following special REXX variables that are modifiable.

Variable Name	Description
RC	The return code from any executed host command. If IMFEXEC detects an error, it sets the REXX TRACE Negative condition to TRUE. As a result, the incorrect IMFEXEC statement is traced. After echoing the incorrect IMFEXEC statement to the BBI-SS PAS Journal, REXX issues its own trace message, which is prefixed by +++.
RESULT	The value of an expression returned by the RETURN command.
SIGL	The line number of the statement currently executing when the last transfer of control to a label took place.

TSO Non-Modifiable Control Variables

The TSO/E REXX language itself does not provide the non-modifiable variables that the CLIST language does. Instead, built-in and external functions are used to obtain the values and assign them to variables. Refer to the section describing “Using TSO/E Functions for REXX EXECs” on page 19 and “Using TSO/E REXX Commands in REXX EXECs” on page 20 in this manual.

Using LOCAL Variables and Pools

This pool is created when an EXEC is assigned to a thread and is deleted when the thread terminates. Variables in this pool are created by:

- Using the IMFEXEC command VPUT to move TSO variables into the LOCAL pool

- Using IMFEXEC CMD with the RESPONSE capability to issue MVS commands (refer to “CMD (MVS Version with Response through X-MCS Consoles)” on page 310) and by invoking SYSPROG services using the IMFEXEC RES command

The LOCAL pool is useful for passing variables between EXECs executing within the same thread (for example, EXEC-initiated EXECs where WAIT(YES) is coded). For example, the calling EXEC includes an IMFEXEC VPUT statement to put variables from the TSO pool into the LOCAL pool. Then, the called EXEC can operate on those variables by using IMFEXEC VGET to get those variables from the LOCAL pool into the TSO pool; for example:

```
N=1                       /* creates a TSO variable in TSO pool*/
"IMFEXEC VPUT N LOCAL"   /* IMFEXEC VPUT cmd places variable in LOCAL pool*/
```

This example stores variables from the TSO pool to the LOCAL pool.

Local variables are not available to EXECs invoked by the REXX CALL function.

Using SHARED Variables and Pools

SHARED variables are a pool of GLOBAL variables maintained in CSA. Variables in this pool are:

- Created by a user who uses the IMFEXEC VPUT statement in an EXEC

  For example:

```
N=1                      /* creates a TSO variable in TSO pool*/
"IMFEXEC VPUT N SHARED"  /* IMFEXEC VPUT cmd places variable in SHARED
                               pool*/
```

  This example stores variables from the TSO pool to the SHARED pool.

- AutoOPERATOR–supplied

  AutoOPERATOR supplies a set of read-only SHARED variables that begin with the prefix Q.

  If you create your own new variables, **do not** use a prefix of Q.

If a BBI-SS PAS warm start is performed: The SHARED variable pool is kept, and all variables have the same values as before the warm start.

If a BBI-SS PAS cold start is performed: The SHARED pool will be reset **only if you specify** the RESET parameter in your BBI-SS PAS JCL. The default is NORESET. Refer to "Cold Start of a BBI-SS PAS" in the *MAINVIEW Administration Guide* for more information on resetting the variable pool at BBI-SS PAS cold start.

You can also reset the pool by issuing the statement:

```
"IMFEXEC VDEL ALL SHARED"
```

in an EXEC. This deletes all the variables from the SHARED pool except the AutoOPERATOR-supplied variables.

To display the contents of the variable pool, use the BBI control command DISPLAY VPOOL (parameters). Refer to the *MAINVIEW Administration Guide* for more information about the BBI control commands.

Serializing Variables

During the time between the VGET and the use of the variable, the value in the SHARED pool may have been modified by another EXEC. EXEC authors are responsible for ensuring variable integrity through the consistent use of ENQ and DEQ facilities throughout the automation procedures.

Refer to "Sharing Variables while Multi-Threading EXECs" on page 70 for more information.

The following lists AutoOPERATOR-supplied SHARED variables that can be used with the IMFEXEC VGET command in an EXEC but cannot be used with IMFEXEC VPUT.

AutoOPERATOR-Supplied SHARED Variables

Variable	Description
QAOREL	Contains a 5-character string indicating the release of AutoOPERATOR The string takes the format **v.r.m** where: **v** Is the version level **r** Is the release level **m** Is the modification level
QIMFID	The BBI subsystem ID of this BBI-SS PAS.
QIMGSTA	(IMS & DB2 Performance Products only) The status of BBI-SS PAS Image logging as ACTIVE or INACTIVE.
QIMGSUF	(IMS & DB2 Performance Products Only) The suffix of the current or last active BBI-SS PAS Image data set. If logging has never been initialized, the value is null.
QIMSID	The IMSID of the IMS/VS being monitored. This IMSID is available only when IMS/VS is active. This IMSID is the same as the IMS/VS identified by QIMSNAME.
QIMSNAME	The jobname of the IMS/VS being monitored by this BBI-SS PAS.
QIMSREL	Contains the IMS release number.
QIMSSTA	The status of IMS/VS (ERE, WARM, COLD, or INACT).
QJNLSTA	The status of BBI-SS PAS Journal logging as ACTIVE or INACTIVE.
QJNLSUF	The suffix of the current or last active BBI-SS PAS Journal data set. If logging has never been initialized, the value is null.
QSMFID	The SMF system ID of the system where the EXEC is running.
QSSNAME	Contains the jobname of the SS address space.

AutoOPERATOR COMMAND/POST Extension Shared Variables

The following are REXX shared variables that are used when performing a GME connection:

Variable	Description
QGMADDR.*GMEID*	IP address of *GMEID* (GME node).
QGMTGTHB.*GMEID*	Target heartbeat interval in minutes that AutoOPERATOR waits before sending another heartbeat to target “gmeid”.
QGMLCLHB.*GMEID*	Local heartbeat interval in minutes between AutoOPERATOR’s receiving of heartbeats from target “gmeid”.
QGMLPORT.*GMEID*	Listener port for the web server. Zero means no listener port is specified.
QGMMSGL.*GMEID*	Maximum length of a message accepted from the GME node.

Variable	Description
QGMNAME.*GMEID*	Host name of the local GME node.
QGMRTC.*GMEID*	Maximum connection retry count for the GME node.
QGMRTI.*GMEID*	Connection retry interval for the GME node in minutes.
QGMSTAT.*GMEID*	Status of *GMEID* (ACT, INACT, or DISCO).
QGMTRAPP.*GMEID*	Minimum level of Application trace records to be sent by *GMEID*.
QGMTRGME.*GMEID*	Minimum level of GME trace records to be sent by *GMEID*.
QGMTRSEC.*GMEID*	Minimum level of Security trace records to be sent by *GMEID*.
QGMWND.*GMEID*	Maximum number of messages that require acknowledgements sent to *GMEID* without waiting for previous messages to be acknowledged. Zero indicates no maximum.

Using the PROFILE Pool

The PROFILE pool is a pool of GLOBAL variables maintained in the extended private area of the BBI-SS PAS and in a checkpoint data set named BBIVARS referred to by the DD statement in the BBI-SS PAS JCL.

Variables in this pool are created by:

- A user who uses the IMFEXEC VPUT statement in an EXEC

  For example:

```
N=1                        /* Creates a TSO variable in TSO pool*/
"IMFEXEC VPUT N PROFILE"   /* IMFEXEC VPUT cmd places variable in
                               PROFILE pool*/
```

  This example stores variables from the TSO pool to the PROFILE pool.

The variables are written to the BBIVARS data set every time the IMFEXEC VCKP command is issued or when an EXEC that updated any PROFILE variable is terminated.

This variable pool is reconstructed from the BBIVARS data set whenever the BBI-SS PAS is restarted. Each variable then contains the value last VPUT into it prior to the BBI-SS PAS termination. Variable integrity is maintained across IPLs and even if the BBI-SS PAS abends, except where:

- The BBI-SS PAS abends after a variable is VPUT to the PROFILE pool but before the EXEC ends
- Before an IMFEXEC VCKP command is issued for the variable

Under these circumstances, the variable in the PROFILE data set or disk would be that of the last completed update.

Serializing Variables

During the time between the VGET and the use of the variable, the value in the PROFILE pool may have been modified by another EXEC. EXEC authors are responsible for ensuring variable integrity through the consistent use of ENQ and DEQ facilities throughout the automation procedures.

Refer to “Sharing Variables while Multi-Threading EXECs” on page 70 for more information.

Saving Data in a Variable Pool

Complex EXECs may use many input sources, such as performance monitors and subsystem messages, to create automation procedures. These procedures can depend on several factors that vary over time. An example of these factors might be the name of the current shift operator or the name of the on-call IMS support person.

Variable pools provide a useful means of saving this type of information for use by several automation procedures.

Potential Use

It is useful to localize site-dependent automation information (such as names and phone numbers of key personnel) in variables for all automation procedures to use. A simple EXEC can be written to set these variables whenever the variable pool is reset.

Describing the Example

This example shows an EXEC that is used to set site-dependent automation information in the PROFILE variable pool.

Information about key personnel is hardcoded in the EXEC (for example: name, user ID, and telephone numbers). The EXEC creates LOCAL variables for this information with the variables:

- NAME
- USERID
- WORKPHON
- HOMEPHON
- PAGER

The EXEC then places the variables into the PROFILE pool under one variable name, IMSPROG.

Example

```
/* REXX */
/*****************************************************************/
/* DOC GROUP(MVS) CODE(J2)                                       */
/* DOC DISP(YES) AUTHOR(B&B)                                     */
/* DOC DESC(SAVING VARIABLES TO PROFILE POOL)                    */
/*****************************************************************/

NAME = 'JOHN_SMITH'
USERID = 'JJH1'
WORKPHON = '800/323-2375'
HOMEPHON = '312/666-1234'
PAGER = '312/999-9999'
"IMFEXEC VDCL IMSPROG LIST(NAME USERID WORKPHON HOMEPHON PAGER)"
"IMFEXEC VPUT IMSPROG PROFILE"
ENDEXIT: END
```

Figure 10. Saving Variables in a Variable Pool

Retrieving Data from a Variable Pool

This section describes how and why you can retrieve information from a variable pool with EXECs.

Potential Use

There are many instances when you might want to create EXECs to notify individuals or groups of individuals about serious operations situations. It is advantageous to create these notifications in a general way so that they refer to a title or a group name, but you can also write an EXEC that notifies specific individuals by name when a situation occurs.

Variable pools provide this capability by allowing you to store variable data such as names and phone numbers and retrieve them later.

Describing the Example

This EXEC retrieves the name, user ID, and telephone numbers of the IMS systems programmer from the PROFILE pool where it was saved in the example on page “Example” on page 68.

The VGET statement retrieves the variable IMSPROG from the PROFILE pool. Because it was saved and retrieved as a list variable, the data is mapped in the variables NAME, USERID, and so on.

The data retrieved from IMSPROG is used to fill in the variable fields needed in the ALERT command. Finally, a return code is set to zero and the EXEC exits.

Example

```
/* REXX EXEC */
/*************************************************************/
/* DOC GROUP(MVS) CODE(J2)                                   */
/* DOC DISP(YES) AUTHOR(B&B)                                 */
/* DOC DESC(RETRIEVING VARIABLES)                            */
/*************************************************************/

"IMFEXEC VDCL IMSPROG LIST(NAME USERID WORKPHON HOMEPHON PAGER)"
"IMFEXEC VGET IMSPROG INTO(IMSPROG) PROFILE"
"IMFEXEC ALERT IMSPROG"TIME()"",
         "'IMSPROG IS NEEDED.  CALL "NAME" AT /N "WORKPHON" OR",
         ""HOMEPHON"'" "FUNCTION(ADD) PRI(MAJOR) QUEUE(IMSPROG)"
"IMFEXEC EXIT CODE(0)"
EXIT
```

Figure 11. Retrieving Variables in a Variable Pool Example

Sharing Variables while Multi-Threading EXECs

If you are allowing concurrent execution of multiple EXECs (see “Multi-Threading EXECs to the Normal or Priority Queue” on page 75), then GLOBAL variables might be accessed and modified by several EXECs concurrently. AutoOPERATOR does not serialize variable usage between IMFEXEC VGET and VPUT commands. You are responsible for the contents of your SHARED or PROFILE pool. The IMFEXEC VENQ and VDEQ statements are provided to serialize any resource. They are especially useful for serializing the use of variables.

Potential Use

You must be careful if a GLOBAL variable can be updated by different EXECs concurrently or if an EXEC that updates a GLOBAL variable executes multiple times concurrently due to the use of multi-threading. This could eventually lead to disastrous results.

This example EXEC updates GLOBAL variables; it uses a locking mechanism provided by the IMFEXEC VENQ command to avoid variable corruption.

Describing the Example

This EXEC serializes a resource named ABENDCNT. The site that uses this EXEC has set a standard saying that GLOBAL variables are serialized using a resource name that is identical to the variable name. All EXECs within the site must conform to the standard or variable integrity might not be maintained.

The EXEC obtains an exclusive ENQ on the resource, reads the variable from the SHARED pool, performs some operations on the variable, saves the variable back in the SHARED pool, releases the resource, and exits.

Example

```
/* REXX EXEC */
/*-------------------------------------------------------------------*/
/* DOC GROUP(AOS) FUNC(AOSAMP)  DESC(USING VENQ AND VDEQ)            */
/* DOC DISP(YES)  AUTHOR(JAC)                                        */
/*-------------------------------------------------------------------*/
"IMFEXEC VENQ 'ABENDCNT' EXC"
IF IMFCC NE 0 THEN EXIT(16)
"IMFEXEC VGET ABENDCNT"
.
.
.
ABENDCNT=ABENDCNT+1
"IMFEXEC VPUT ABENDCNT"
"IMFEXEC VDEQ 'ABENDCNT'"
IF IMFCC NE 0 THEN EXIT(16)
"IMFEXEC EXIT CODE(0)"
```

Figure 12. Using VENQ and VDEQ to Serialize Variables

Rule-Initiated EXECs Initiated by MVS Multi-Line or Multi-Segment Messages

Rule-initiated EXECs fired by multi-line WTOs or multi-segment message can access only the **first** line or segment of the MVS or IMS message with symbolic parameters on the PROC statement. For more information about Rule-initiated EXECs, refer to “Rule-Initiated REXX EXECs” on page 29.

To access the additional lines and segments in the MVS or IMS message, the EXEC must use the IMFEXEC VGET statement to create LOCAL variables for LINE1 through LINExxxx (depending on the number of lines of the WTO).

The actual number of lines or segments in the MVS or IMS message is stored in the TSO variable IMFNOL. If you have five lines, then IMFNOL=5.

Potential Use

This section describes how to handle accessing the additional lines of information from multi-line WTOs or multi-segment messages.

This example shows an MVS multi-line WTO that fired a Rule-initiated EXEC:

```
JOB01766 IEF450I JDB1ABND - ABEND=S0C1 U0000 REASON=00000001 984
     984         TIME=10.51.34
```

Describing the Example

In this example, the ARG statement does not contain any symbolic parameters because the first line of the message is retrieved from the LINE01 variable.

However, in general, the first line could also be retrieved using symbolic parameters (such as in Rule-initiated EXECs). This example demonstrates this process. The EXEC retrieves all lines of IEF450I and writes the output of this message to the BBI-SS PAS Journal.

Example

```
/* REXX EXEC */
/*****************************************************************/
/* DOC GROUP(AOS) FUNC(AOSAMP)  DESC(RETRIEVING MULTILINE WTO)    */
/* DOC DISP(YES)  AUTHOR(JAC)                                     */
/*****************************************************************/

 DO I = 1 to IMFNOL
   "IMFEXEC VGET LINE"I" LOCAL"
   "IMFEXEC MSG '"VALUE('LINE'I)"'"
 END
```

Figure 13. Multi-Line WTO EXEC Example

Chapter 5. Controlling EXEC Execution

This chapter discusses how to:

- Schedule EXECs to be run
- Schedule an EXEC that waits for another EXEC to complete (synchronous execution)
- Invoke an EXEC
- Monitor and control EXEC execution using BBI control commands
- Code an EXEC to display CPU consumption

Scheduling EXECs

Each EXEC represents a unit of work that needs to be completed. Just as any system that handles requests to complete work, AutoOPERATOR provides scheduling facilities for EXECs. EXECS are queued for execution to either:

- The Normal queue
- The Priority queue

Defining Threads

When an EXEC is scheduled to either the Normal or Priority queue, it waits for a server, called a **thread**, to become available. The number of threads available to the Normal and Priority queues are defined by the installation (see “Multi-Threading EXECs to the Normal or Priority Queue” on page 75).

An EXEC remains assigned to a single thread until the EXEC terminates. In a single thread, only one EXEC can be actively running at any one time. Multiple EXECs can execute under the same thread: this is called **synchronous execution**. Refer to “Invoking EXECs Synchronously with IMFEXEC SELECT(EXEC) WAIT(YES)” on page 78 for more information.

Scheduling EXECs to the Normal Queue

By default, all EXECs (ALERT-initiated, Time-initiated, and so on) are scheduled through the Normal queue regardless of how they are invoked. The EXEC executes immediately if there is a thread available, otherwise it waits until one becomes available. The default setting is one thread for the Normal queue.

Scheduling EXECs to the Priority Queue

The Priority queue is for EXECs that must not wait for a long backlog of processing. To send an EXEC to the Priority queue, you must identify the EXEC in either of two ways:

- Specify the name of the EXEC in BBPARM member AAOEXP00
- Use the PRI(HI) parameter of the IMFEXEC SELECT command

  Refer to "SELECT" on page 351 for more information about how to code the IMFEXEC SELECT command.

Both these methods are described in this chapter.

Naming the EXEC in BBPARM member AAOEXP00: In BBPARM member AAOEXP00, the EXEC= parameter allows you to specify the names of EXECs that will automatically receive high priority status. The *MAINVIEW AutoOPERATOR Customization Guide* contains information for BBPARM member AAOEXP00.

Example 1: BBPARM member AAOEXP00 contains the statement:

```
EXEC=THREE
```

This parameter specifies that an EXEC named `THREE` is queued to the Priority queue whenever it is invoked and regardless of how it is invoked (for example, Rule-initiated, ALERT-initiated, and so on).

The exception to this situation is for EXEC-initiated EXECs where an EXEC is invoked with the IMFEXEC SELECT statement. See Example 2 on this page for clarification.

Example 2: If you use the IMFEXEC SELECT statement to schedule an EXEC that is named in BBPARM MEMBER AAOEXP00, you must still code the parameter PRI(HI) to have the EXEC scheduled to the Priority queue.

To schedule an EXEC named in BBPARM member AAOEXP00 with IMFEXEC SELECT, code:

```
IMFEXEC SELECT EXEC(THREE XYZ1 XYZ2) PRI(HI)
```

EXEC `THREE` executes immediately on the Priority queue if there is at least one thread available. If there is no Priority thread available, then the EXEC waits until a Priority thread is available.

You must restart the BBI-SS PAS to pick up new EXEC names added to AAOEXP00.

Using the IMFEXEC SELECT Statement and the PRI(HI) Parameter: You can use the PRI(HI) parameter with the IMFEXEC statement to schedule an EXEC to the Priority Queue. To do this, code the PRI(HI) operand on the IMFEXEC SELECT command that calls the EXEC.

For example; the following statement:

```
IMFEXEC SELECT EXEC(FOUR XYZ1 XYZ2) PRI(HI)
```

schedules EXEC `FOUR` for execution on the Priority queue.

EXEC `FOUR` executes immediately if there is at least one Priority thread available. If no Priority thread is available, then the EXEC waits until a Priority thread is available.

EXECs can be added dynamically to libraries in the SYSPROC concatenation so you do not have to restart the BBI-SS PAS if you use this method to schedule priority EXECs but you must issue the `.RESET BLDL SYSPROC` command.

Multi-Threading EXECs to the Normal or Priority Queue

Define multiple threads for the Normal queue and the Priority queues in the BBPARM member AAOEXP00. This allows concurrent execution of multiple EXECs. The following table shows how to do this.

Queue Name	Parameter Name	Example
Normal queue	MAXNORM= Specify the number of threads on the MAXNORM= statement in BBPARM member AAOEXP00	For example, the parameter statement: `MAXNORM=10` defines 10 threads for the Normal queue and 10 EXECs can run concurrently in the Normal queue.
Priority queue	MAXHIGH= Specify the number of threads on the MAXHIGH= statement in BBPARM member AAOEXP00	For example, the parameter statement: `MAXHIGH=5` defines five threads for the Priority queue and five EXECs can run concurrently in the Priority queue.

If you are operating with MAXNORM or MAXHIGH set to greater than one and then want to reset MAXNORM=1, you must ensure that no automation procedures are dependent on the concurrent execution of several EXECs.

CAUTION:
Multi-threading EXECs requires additional virtual storage in the BBI-SS PAS address space. If virtual storage is insufficient, the SS will fail with an x78 abend.

Multi-threading EXECs may also require variable serialization using ENQ/DEQ logic. Refer to “Sharing Variables while Multi-Threading EXECs” on page 70 for more information.

Using EXEC Threads and Their Effect on Performance

BMC Software recommends that all automation be done within a Rule (or set of Rules) whenever possible. This is both for performance and storage considerations. Rules are faster and use less resources. However, not all automation can be done within Rules. The following information and/or recommendations offers assistance in tuning your automation for use with EXECs.

AutoOPERATOR is shipped with the following default values for MAXHIGH and MAXNORM EXEC threads. When installed with AutoCustomization:

- MAXNORM=5
- MAXHIGH=5

When using BBPARM member AAOEXP00 as it is shipped with AutoOPERATOR, the settings are:

- MAXNORM=1
- MAXHIGH=5

It is necessary to understand of the two types of EXEC threads, MAXNORM and MAXHIGH. EXECs are normally considered batch work. This batch work may occasionally get backed up. You can control the maximum number of queued EXECs with the MAXNORMQ and MAXHIGHQ fields in BBPARM member AAOEXP00.

Because not all automation can be done with Rules, AutoOPERATOR provides a way of scheduling higher priority automation within an EXEC. This is where the Priority EXEC thread comes into use. AutoOPERATOR intends that the Priority queue does not get backed up (or it should back up much less). Therefore, the default MAXHIGH value shipped in the sample BBPARM member AAOEXP00 is much higher than the value for Normal EXECs (MAXNORM).

Note: When tuning automation through EXECs, you should note that the actual dispatching priority of Priority EXECs is the same as a Normal EXEC.

These Priority EXECs compete for CPU on the same dispatching priority as NORM EXECs. The concept of a Normal and Priority EXEC queue is designed as a method to have 2 queues, where one is used less and therefore, scheduled faster.

For example, if you have MAXNORM=5 and MAXHIGH=5 and currently have 10 Normal EXECs scheduled, you would have 5 currently running and 5 queued up to run. If you then want to schedule a new EXEC, if it was Normal it would be queued up behind the other 5, but if it was Priority, it runs immediately.

It is also important to know that “more is not faster”. Using more EXEC threads means more tasks for MVS to manage. The CPU overhead goes up because there are more EXEC threads. Each system is different and no specific value for CPU consumption (or optimum number of EXEC threads) can be provided. For most sites the default value of MAXNORM=5 and MAXHIGH=5 is sufficient. However, the optimum value for an individual system may be lower or higher.

Additional Recommendations

Ypu should take into consider the following when tuning AutoOPERATOR for the optimum value of MAXNORM and MAXHIGH:

1. Start with the least number of EXEC threads needed to get the desired throughput.
2. Use MAXNORMQ and MAXHIGHQ along with the warning settings in BBPARM member AAOEXP00 so you can be advised when the EXEC threads queue up. Adjust the MAXNORM and MAXHIGH values as needed.
3. All EXEC threads, whether CLIST or REXX EXECs, use a large amount of private storage below the line. Use of an excessive amount of EXEC threads will cause LSQA to limit the amount of low private available to the system.
4. Carefully consider the actions within the EXECs before changing MAXNORM and/or MAXHIGH settings. Move automation out of EXECs to Rules whenever possible. Rules are always faster and always use less system resources.
5. Consider breaking EXECs that wait for an excessive amount of time into multiple parts. A combination of Rules and EXECs may be used to replace one long running EXECs.

Each system and automation strategy is different and tuning should be done on each system. However, where you need some recommendations to start with, you can also”

1. Start with the default values supplied in AAOEXP00.
2. Use the following threshold control fields in AAOEXP00 to determine when you have a problem.

   MAXNORMQ=0 (default of 0 means not in use)

   MAXHIGHQ=0 (default of 0 means not in use)

   WARNLVL1=60 (default of 60 but not valid until MAXNORM or MAXHIGH used)

   WARNLVL2=75 (default of 75 but not valid until MAXNORM or MAXHIGH used)
3. Only change automation strategy after careful analysis of what is causing the queues to back up. Remember, more EXEC threads use more CPU and therefore may increase the queue back log.
4. Use the least number of EXEC threads needed to accomplish the required throughput.
5. MAXHIGH should be set equal or higher to MAXNORM.
6. Lastly, more EXEC threads means higher use of LSQA, since each thread needs a MVS TCB, etc. which all reside in LSQA. If you have been experiencing a shortage of low private storage (for example, ABENDS s878-10), check the values of MAXHIGH and MAXNORM.

Any value greater than the recommended value of 5 and 5 respectively should be carefully considered as a possibility of contributing to a shortage of low private storage.

Invoking EXECs Synchronously with IMFEXEC SELECT(EXEC) WAIT(YES)

Some automation procedures may need to include more than one EXEC to run. Using the IMFEXEC SELECT statement in an EXEC allows one EXEC to invoke another EXEC-initiated EXECs are usually subroutines or service routines that carry out specialized tasks needed by several automation procedures.

An EXEC can invoke another EXEC under the same thread (synchronously) or under a new thread (asynchronously). IMFEXEC SELECT allows one EXEC to invoke another. If IMFEXEC SELECT is coded with WAIT(YES), the called EXEC is invoked to execute under the same thread. Otherwise, the called EXEC executes as a separate task under a new thread.

The following table shows where you can find more information.

To read about...	See...
EXEC-initiated EXECs	“EXEC-Initiated REXX EXECs” on page 42
Using the IMFEXEC SELECT statement and its parameters	“SELECT” on page 351
Using variables	“Using Variables in REXX EXECs” on page 49

Passing Control of the EXEC

By specifying the WAIT(YES) parameter on an IMFEXEC SELECT statement, an EXEC can schedule another EXEC, wait for its completion, and then resume execution.

When an EXEC invokes another EXEC using the WAIT(YES) parameter, control is passed immediately to the called EXEC. The called EXEC can use the IMFEXEC statements VDCL, VGET, and VPUT to access all the LOCAL, GLOBAL, and SHARED variables created by the first EXEC, but it does not have access to any of the TSO variables created by the first EXEC.

The execution of the calling EXEC is suspended when the called EXEC is being processed. When the called EXEC terminates, the first EXEC receives control at the first statement immediately after the IMFEXEC SELECT statement.

BBI variables IMFCC and IMFRC are used to report the success of the scheduled WAIT(YES) EXEC. See “Understanding Completion Codes for EXEC-Initiated EXECs with WAIT(YES) and User Written Programs” on page 354 for a complete discussion.

For EXECs invoked with the IMFEXEC SELECT EXEC( ) WAIT(yes) statement, the two ways to pass back results are using:

- IMFEXEC EXIT CODE(x)
- A local, shared, or profile variable

Using RETURN will give control back to the calling EXEC but the passed back value (RESULT) is not supported.

Implementing an EXEC

Once an EXEC has been designed, coded, and tested, it can be implemented in AutoOPERATOR using two steps:

- Move the EXEC to a data set in the SYSPROC or the SYSEXEC library concatenation of your production BBI-SS PAS.

  If you use both the SYSPROC and SYSEXEC members, the following limitations apply:

  – EXECs in SYSEXEC can be invoked **only if REXX=YES is specified** in BBPARM member AAOEXPxx, where xx is the suffix of the member being used. SYSEXEC can contain only REXX EXECs and tokenized REXX EXECs.

    Refer to the *MAINVIEW AutoOPERATOR Customization Guide* for more information about setting REXX=YES in BBPARM member AAOEXP00.

  – If you have an EXEC with the same name in both the SYSPROC and SYSEXEC members, then the EXEC is SYSEXEC is executed.

  – If you have an EXEC with the same name in both the SYSPROC and SYSEXEC members, then disabling the EXEC in one member also disables it in the other member.

    In other words, the EXEC always has the same status, no matter which member it is in.

  – Both SYSPROC and SYSEXEC can be browsed from the EXEC Management application.

  – If you try to invoke an EXEC from the EXEC Management application that is listed in SYSPROC and is **also listed in** SYSEXEC, you will receive an error message.

- If the EXEC was moved to the BBPROC library concatenation (the DDNAME is SYSPROC) after the BBI-SS PAS was recycled, issue the command:

  ```
  .RESET BLDL SYSPROC
  ```

  and the EXEC will be available immediately. Changes to existing EXECs take effect immediately without the `.RESET command` but new EXEC names cannot be accessed until the .RESET command is issued or the SS is started.

  CAUTION:
  If you try to access new EXEC names without a SS restart or resetting, you will receive the following error message displayed in the upper corner:

  ```
  EXEC NOT FOUND
  ```

Controlling EXEC Execution

This section describes how you can control the execution of EXECs once they are invoked by:

- Setting time and CPU limits for EXECs
- Displaying the status of an EXEC
- Stopping (disabling), starting (enabling), and cancelling an EXEC

  These functions are performed using the BBI control commands. Refer to the *MAINVIEW Administration Guide* for more complete information about the BBI control commands.

Setting Time and CPU Limits for EXECs

The following list describes how to set CPU and time limits for EXECs.

- Set the parameters in BBPARM member AAOEXP00:
  - PEREXLIM
  - TIMEXLIM

  to control time and/or CPU limits for all EXECs.
- Use the IMFEXEC CNTL statement and its parameters in an EXEC:
  - PERLIM(xx)
  - TIMLIM(xx)

  to control time and CPU limits for a specific EXEC.

  If these parameters are specified in an EXEC, they override the parameters set on PEREXLIM and TIMEXLIM in BBPARM member AAOEXP00. Refer to “CNTL” on page 321 for a complete description of IMFEXEC CNTL and its parameters.

  PERLIM(xx)

  For example, if you specify:

  ```
  IMFEXEC CNTL PERLIM(15)
  ```

  The EXEC will run until it exceeds 15% of the CPU during any 15 second interval. If the EXEC exceeds 15%, it is automatically terminated.

  TIMLIM(xx)

  For example, if you specify:

  ```
  IMFEXEC CNTL TIMLIM(10)
  ```

  the EXEC will run until it exceeds 10 CPU seconds. If the EXEC exceeds 10 CPU seconds, it is automatically terminated and abend message U3001 is issued.

  When an EXEC exceeds the limits you set, check to see if it is executing correctly or if it has gone into a loop. Use the EXEC Management Application to determine if EXECs are running closely to the limits you have set. BMC Software recommends that you set these parameters with non-zero values **because a value of zero allows unlimited CPU consumption by an EXEC**.

Displaying EXEC Execution Status

You can monitor and control the progress of an EXEC by using the BBI control command DISPLAY EXEC in the BBI Log display. The format of the command is:

```
.DISPLAY E|EXEC ALL|HIGH|NORMAL|STATS
```

This command shows the statistics for all running and queued EXECs. By examining the progress of an EXEC, you can decide whether you need to take actions such as terminating or disabling the EXEC.

You also can use the EXPAND primary command on the EXEC Management application to display currently active EXECs. For more information and an example, refer to the chapter "Managing EXECs Using the EXEC Management Application" in the *MAINVIEW AutoOPERATOR Basic Automation Guide*.

Cancelling, Stopping, and Starting EXEC Execution

You might decide to manually control the progress of an EXEC once it has been invoked. By using the BBI control command .DISPLAY, you can see the progress in the BBI Log display. If you decide to intervene in the EXEC, you can use the following BBI control commands:

BBI Control Command	**Action taken**
.CANCEL	Terminates the execution of an EXEC while it is running or if it is waiting for a thread to become available.
.STOP	Disables an EXEC that is running. This command prevents the EXEC from being invoked again until it is either STARTed by the BBI START command or RESET by the BBI RESET BLDL SYSPROC command. Does not cancel the current EXEC.
.START	Enables an EXEC that has been STOPped and makes it available to be invoked. This command does **not** invoke an EXEC.

You can also control the execution of an EXEC with the EXEC Manager application. Refer to the *MAINVIEW AutoOPERATOR Basic Automation Guide* for more information.

Analyzing EXEC Performance Using the EXEC Management Application

This section discusses how you can use the EXEC Management Application to analyze how well EXECs are running on your system.

For a more general discussion about the AutoOPERATOR EXEC Management Application, refer to the chapter “Using the EXEC Management Application” in the *MAINVIEW AutoOPERATOR Basic Automation Guide*.

The EXEC Management Application has panel displays that show EXEC usage statistics such as:

- The highest CPU total
- The average CPU percentage
- The number of times an EXEC as been executed since the last AutoOPERATOR subsystem cold start

For performance analysis, the following data columns are of special interest:

Column Heading	Description
SCHED	Is the number of times the EXEC has been scheduled. Each time an EXEC is scheduled from a Rule, ALERT-initiated EXEC, external program, the TS command line, or another EXEC, the SCHED count is incremented. A REXX program executed through a CALL statement is not counted. When EXECA calls EXECB (with an IMFEXEC SELECT statement where WAIT(YES) is specified), both EXECA and EXECB are counted in the SCHED count.
TOTCPU	Is the sum of CPU time used for all scheduled executions of the EXEC since the SS was started. If the EXEC schedules another EXEC (with an IMFEXEC SELECT statement where WAIT(YES) is specified), then CPU collection for the first EXEC is suspended until the selected EXEC returns control. If a REXX EXEC executes another REXX EXEC using the REXX CALL facility, the CPU time is charged to the calling EXEC.
AVGCPU	Is the value when the value in the TOT-CPU column is divided by the value in the SCHED column.
MAXCPU	Is the greatest amount of CPU time the EXEC used during any single execution.

Using the SORT Command in the EXEC-Management Application

The SORT command can be used to categorize EXECs by their performance.

To use this command, enter `SORT` on the command line of the EXEC Management panel. For example:

```
SORT AVGCPU D
```

sorts the display where the EXECs with the highest average CPU consumption are shown at the top.

By sorting the display, you can more easily see where the AVERAGE CPU consumption of a specific EXEC is equal-to or less-than the limit set in the TIMEXLIM parameter for the subsystem.

You should address the EXECs that are executing above this limit for tuning.

Key Performance Indicator Discrepancies: Other discrepancies that can (and should) be analyzed are:

- When (for any EXEC) the MAXCPU is at least 25% greater than the AVGCPU column

  This indicates that an EXEC may be subject to spikes in CPU consumption. This may be due to the volume of its input or other events that drive the EXEC.

- When the SCHED value (the number of times scheduled) is incrementing rapidly

  This can indicate a scheduling loop or a flood of message events.

  Note that in this event, the TOTCPU, AVGCPU, and MAXCPU numbers may be low. Generally, EXECs that are being initiated excessively are Rule-initiated EXECs that are scheduled by a flood of events.

  Often, such problems are resolved by altering the design of the Rule-initiated EXEC.

Writing EXECs that Display CPU Consumption

A common problem with EXEC performance is an EXEC exceeding the CPU thresholds set for AutoOPERATOR. The resulting abend can be bypassed by using IMFEXEC CNTL statement in the EXEC to reset the limits. However, this can potentially expose your system to excessive CPU consumption and/or program loops, and diagnosing a runaway situation such as this is difficult.

One technique for diagnosing these problems involves writing some additional code in the EXEC to monitor itself.

For example:

- When writing REXX EXECs, use the statement:

  ```
  TSO FUNCTION "SYSVAR('SYSCPU')"
  ```

  This returns the total amount of CPU seconds used to date for the TCB on which the EXEC is running.

- Change the EXEC to set a control variable with the CPU value on entry

  The control variable can then be manipulated later as required.

For example:

```
/* REXX */
parse arg exec_name p1 .
do x = 1 to p1 by 1
   "VGET VARNAME"||x "SHARED"
   "MSG 'VARNAME"||x "=" value("VARNAME"||x)"'"

end x
"EXIT CODE(0)"
exit 0
```

This EXEC is a subroutine that displays an array of variables from the SHARED variable pool on the sub-system journal. Occasionally, it may spike in CPU consumption because the number of array items spikes. However, this is not a situation that can be seen and analyzed from the EXEC Management application.

Therefore, you can modify the EXEC to identify the problem and display diagnostic data using the SYSCPU function as shown:

```
/* REXX */
entry_cpu = trunc(SYSVAR('SYSCPU')) /* Get CPU Time on entry */
parse arg exec_name p1 .
do x = 1 to p1 by 1
   "VGET VARNAME"||x "SHARED"
   "MSG 'VARNAME"||x "=" value("VARNAME"||x)
   time_used = trunc(SYSVAR('SYSCPU')) - entry_cpu

      if time_used => "CPU LIMIT SET ON THE SYSTEM" then
      do
        "ALERT" exec_name"@CPU 'CPU TIME AT" x "ELEMENTS IS"
      time_used"'"
        entry_cpu = trunc(SYSVAR('SYSCPU'))
      end

end x
"EXIT CODE(0)"
exit 0
```

In this example, the EXEC itself does some preliminary analysis for the EXEC writer. More typically, this routine would be built into a common function which can be called.

BBSAMP member AOXCPUFI contains an example of REXX internal functions that you can easily incorporate into another EXEC to selectively call for analysis.

BBSAMP member AOXCPUSI contains an example of a REXX EXEC that uses AOXCPUFI (internally called by a REXX EXEC as CPU_FUNC) to monitor CPU utilization. This example EXEC checks CPU consumption after a defined numbers of operations in its calling routine to determine the threshold number of events that equal a predefined amount of CPU seconds.

Note: BBSAMP member AOXCPUST contains the tokenized version of AOXCPUSI. Refer to “REXX EXEC Considerations” on page 103 for more information about tokenized REXX EXECs.

Chapter 6. Using Advanced Techniques with AutoOPERATOR EXECs

This chapter describes some advanced functions of AutoOPERATOR EXECs and describes how to handle EXECs across more than one BBI-SS PAS and target. Topics include:

- Scheduling EXECs across BBI-SS PASs
- Determining the origin of an EXEC
- Using the program called IMFSUBEX to invoke EXECs from outside AutoOPERATOR
- Testing EXECs
- Deleting, reading, and writing SHARED and PROFILE variables across BBI-SS PASs

Overview

Any BBI-SS PAS address space can monitor other target systems. You can also have multiple BBI-SS PAS address spaces communicating with one another. A target can be:

- Any CICS, IMS, DB2, or MVS system
- Any MVS subsystem

Define these two types of targets as follows:

Target type	BBPARM member	Parameter name
MVS, CICS, IMS	BBIJNT00	target=
MVS subsystem	BBINOD00	subsys=

Because EXECs can interact with any target you specify, careful managing of your EXECs across more than one target or BBI-SS PAS becomes very important. For more information about targets and BBI-SS PAS to BBI-SS PAS communication, refer to the *MAINVIEW Administration Guide*.

Scheduling Messages and EXECs Across BBI-SS PASs

The following items can be sent from one BBI-SS PAS to any other BBI-SS PAS or target, even in remote locations:

- Messages
- EXECs
- ALERTs
- IMF or MAINVIEW for DB2 service commands

This means a single BBI-SS PAS can monitor and control many systems as long as the target systems have a BBI-SS PAS product installed. It is also possible to detect and correct conditions that arise in one target system but affect another target system. This is important because you want to be able to manage and control all the activity between BBI-SS PASs and targets.

To accomplish these tasks, you would use the appropriate IMFEXEC statement and specify the target with the TARGET keyword. The following table shows what tasks you can accomplish and which IMFEXEC statements to use.

Task	**IMFEXEC statement**
Send a message to another target	IMFEXEC MSG TARGET(tgtname)
Send an EXEC to another target	Either: • IMFEXEC SELECT(execname) TARGET(tgtname) • IMFEXEC SET REQ=CALLX This IMFEXEC statement allows access to the timer facility to invoke a time-initiated EXEC. The TARGET keyword allows you to specify another target.
Send an ALERT to another target	IMFEXEC ALERT TARGET(tgtname)
Send an IMF or MainView for DB2 command to another target	IMFEXEC IMFC TARGET(tgtname)
Note: You can also schedule an EXEC to run at another target with the program IMFSUBEX. Refer to "Invoking REXX EXECs from Outside of AutoOPERATOR with IMFSUBEX" on page 93 for more information.	

Refer to "Using the IMFEXEC Statements" on page 237 for the complete description of these IMFEXEC statements.

The target that you specify on these commands must be defined in BBPARM member BBIJNT00 on the local BBI-SS PAS. For information about how to define targets to a BBI-SS PAS, refer to "Step 20: (Required) Define BBI-SS PAS Suffixes and Target System Parameters" in the *MAINVIEW Common Customization Guide*.

These examples show how you can schedule EXECs, messages, ALERTs, and other commands to targets with the TARGET keyword with the appropriate IMFEXEC statement.

Examples

To send a message from one BBI-SS PAS to another target: Use the IMFEXEC MSG statement in an EXEC with the TARGET keyword and specify the name of a target that is defined in BBPARM member BBIJNT00 for an MVS, CICS, IMS, or DB2 system or BBPARM member BBINOD00 for an SS.

The message will be logged on the remote BBI-SS PAS Journal, and no entry will be made on the originating system's Journal. For example:

```
"IMFEXEC MSG 'MANUFACTURING DATABASE IS OFFLINE' TARGET(CICSPROD)"
```

sends a message from a local BBI-SS PAS to the BBI-SS PAS Journal of the production BBI-SS PAS that is monitoring a CICS system called `CICSPROD`.

To schedule an EXEC from one BBI-SS PAS to another target: Use the IMFEXEC SELECT command in an EXEC with the TARGET keyword and specify the name of a target that is defined in BBPARM member BBIJNT00 for an MVS, CICS, IMS, or DB2 system or BBPARM member BBINOD00 for an SS. For example:

```
"IMFEXEC SELECT EXEC(PAYROLL START) TARGET(CICSPROD)"
```

schedules an EXEC from the local BBI-SS PAS to the BBI-SS PAS where the remote CICS production system is defined.

To send a time-initiated EXEC from one BBI-SS PAS to another target: Use the IMFEXEC SET REQ=CALLX statement in an EXEC with the TARGET keyword and specify the name of a target that is defined in BBPARM member BBIJNT00 for an MVS, CICS, IMS, or DB2 system or BBPARM member BBINOD00 for an SS. For example:

```
"IMFEXEC IMFC SET REQ=CALLX @HOURLY START=6:00:00 STOP=20:00:00",
 "I=02:00:00 TARGET(BBSYSA)"
```

schedules an EXEC named `@HOURLY` to be run at two hour intervals beginning at 6:00 am and ending at 8:00 pm on the target system called BBSYSA.

To send an ALERT from one BBI-SS PAS to another target: Use the IMFEXEC ALERT statement in an EXEC with the TARGET keyword and specify the name of a target that is defined in BBPARM member BBIJNT00 for an MVS, CICS, IMS, or DB2 system or BBPARM member BBINOD00 for an SS. For example:

```
"IMFEXEC ALERT NETW2",
 "'COMMUNICATION LINES DOWN:  /N   - DALLAS /N +  - CHICAGO'
FUNCTION",
 "(ADD) QUEUE(NETWORK)",
 "PRIORITY(CRITICAL) COLOR(PINK) TARGET(NYCSYS)"
```

sends a multi-line ALERT to a target called NYCSYS.

To send an IMF or MainView for DB2 command from one BBI-SS PAS to another target: Use the IMFEXEC IMFC statement in an EXEC with the TARGET keyword and specify the name of a target that is defined in BBPARM member BBIJNT00 for an MVS, CICS, IMS, or DB2 system or BBPARM member BBINOD00 for an SS. For example:

```
"IMFEXEC IMFC PLOT ARVTR ABC IMSNAME=PRODIMS TARGET(SYSA1)"
"IMFEXEC IMFC PLOT CSAUT IMSNAME=IMSP TARGET(SYSA1)"
"IMFEXEC IMFC STAT IMSNAME=IMSP TARGET(SYSA1)"
```

invokes synchronous analyzer services such as STAT, CLASQ, or PLOT for automatic logging on a target called SYSA1.

Determining the Origin of a Command or EXEC

The flexibility of the AutoOPERATOR EXEC processor allows an EXEC to be initiated in many ways from many targets. Because commands and EXECs can be issued from one target in one BBI-SS PAS to other targets within or across BBI-SS PASs, you need to know how to determine the origin of a command or EXEC.

Determining the origin of an EXEC is especially important for security reasons because an EXEC can send a message or another EXEC to execute some action on another target. The target should be able to take or not take the action based on the origin of the sending target.

This means that the author of an EXEC must take special steps to ensure that the EXEC's action is appropriate for the situation. AutoOPERATOR provides two variables that allow an EXEC to determine:

- The name of the originating BBI-SS PAS (IMFORGSS)
- The origin of the EXEC (IMFORGN)

  The origin of commands within an EXEC is the same origin as that of the EXEC.

These variables are defined in “TSO Variables Supplied by AutoOPERATOR” on page 54.

Use these variables to determine, for example, if the caller is authorized to execute the EXEC or to determine the user ID that is to receive any informational messages returned from the EXEC. Both variables are automatically available to all EXECs.

Determining IMFORGN

The following table shows what origin (IMFORGN) is, depending on how the EXEC was initially triggered:

If the command or EXEC is:	**Then origin (IMFORGN) is:**
User-initiated (from a BBI-TS)	The user's USERID
Time-initiated	The BBI-SS PAS ID of the BBI-SS PAS that called the EXEC
BBI-SS PAS message-initiated	The BBI-SS PAS ID of the BBI-SS PAS that issued the message
Externally initiated	One of these: • JOBNAME • The RACF user ID Refer to “Invoking REXX EXECs from Outside of AutoOPERATOR with IMFSUBEX” on page 93 for more information about the origin for externally initiated EXECs. See the definition of “ORIGIN”.
IMS message-initiated	The IMS JOBNAME of the calling EXEC
IMS command from an IMS terminal	The LTERM of the IMS terminal
CICS exception-message initiated	The name of the CICS region for which the message was issued
CICS TD-message initiated	The name of the CICS region for which the message was issued

If the command or EXEC is:	Then origin (IMFORGN) is:
DB2 exception-message initiated	The name of the DB2 region for which the message was issued
MVS message-initiated	One of these: • JOB name • STC name • TSO name
EXEC-INITIATED	The EXEC name of the calling EXEC
ALERT follow-up	Either the user ID of the terminal session user or the value of ORIGIN

Example - Determining the Origin of a User-Initiated EXEC

Scenario: For this example, there are two BBI-SS PASs called CICM and CICP. CICP has a CICS target system called CICPROD defined to it.

A BBI-TS user with user ID TSOUSR1 logs onto CICM and schedules an EXEC named PAYROLL. The origin of the PAYROLL EXEC is TSOUSR1.

The PAYROLL EXEC may try to schedule another EXEC, called DATAB, to the CICP subsystem which is monitoring CICPROD. The origin of PAYROLL is CICM (the originating BBI-SS PAS of the EXEC) and it is passed to CICP.

Now, CICP must be able to determine if the origin called CICM is authorized to invoke the DATAB EXEC by searching BBPARM for the authorization member and validating the authority of CICM to run the EXEC.

Invoking REXX EXECs from Outside of AutoOPERATOR with IMFSUBEX

EXECs can be invoked from any job running on a processor with a local BBI-SS PAS address space or from a remote processor. This can be useful to signal an event, such as the completion of an SMF Dump job running in the background. To do this, an EXEC can be invoked from a batch program running as a separate job step or from a callable subroutine within another job or from TSO.

Use the AutoOPERATOR-supplied program called IMFSUBEX to submit these kinds of EXECs. Keyword parameters passed to IMFSUBEX must specify:

- A local BBI-SS PAS Address Space ID (ASID) or an asterisk (*)
- The name of the EXEC to be invoked
- Any operands to be passed to the EXEC

and optionally:

- A different target

Example of a Parameter String Passed to IMFSUBEX

The following parameter string shows a complete example of all the keyword parameters that can be passed to IMFSUBEX:

```
SS(subsys) EXEC(execname p1...pn) [TARGET(tgt)    +
     ORIGIN(source) WAIT(YES)  MSGLVLI(NONE)]
```

The parameters from this statement are described in the following table:.

Keyword	Required/ Optional	Description
SS	Required	Defines a BBI-SS PAS on the same processor as the invoking job. This BBI-SS PAS initially receives and processes the request, sending it to SS(*) or another BBI-SS PAS if TARGET is specified. If specified as *, any BBI-SS PAS found on that processor is acceptable (from one to four asterisks accepted). Also, a generic name can be given by using positional (+) or generic (*) qualifiers, such as SS(+++P) or SS(P*).
EXEC|E	Required	Specifies the name of the EXEC and any parameters to be passed to the symbolic variables defined as input in the EXEC.

Keyword	Required/ Optional	Description
TARGET|T	Required	Identifies a different target from the target system where the EXEC will be invoked. The specified TARGET should match a TARGET=(tgtname) parameter in member BBIJNT00 of BBPARM. The EXEC is scheduled on the subsystem that corresponds to the subsystem specified by the SS parameter. The specified TARGET may also be an SSID which the original subsystem communicates with. If you omit the TARGET keyword, the default is the SSID name of the subsystem that services this IMFSUBEX request. Then, you would not need to specify a TARGET=SS in BBIJNT00. In this case, the AUTHJOB= parameter of your BBPARM authorization member must be specified so that the SSID is recognized as a valid target. For example, you can specify the parameter as: `AUTHJOB=*` in the BBPARM authorization member. With an asterisk, the IMFSUBEX TARGET(). parameter can contain *any* target specified in the BBPARM BBIJNT00 member for the SS() specified subsystem.
ORIGIN|O	Optional	Specifies the source of the origin identifier used for security checking. The default for this parameter is JOBNAME. The following values are valid: • JOBNAME causes the jobname to be used as the security token. • RACF causes the value supplied in the USER= keyword of the job card to be used for the security token. • USER causes the value supplied in the USER= keyword of the job card to be used for the security token. If RACF or USER is specified, IMFSUBEX checks for the existence of the RACF ACEEUSRI for the address space and uses what is specified as the security token. If RACF ACEEUSRI does not exist, the JCTUSER field from the job control table (JCT) is used.
WAIT|W	Optional	Specifies that at completion of the EXEC, either the generated return code of the EXEC or the condition code in batch is passed back from IMFSUBEX. You must use caution when using the TARGET keyword with WAIT. The TARGET keyword reserves the VTAM link between the originating BBI-SS PAS and the target BBI-SS PAS for the duration of the EXEC and accepts no other requests (such as a user wanting to display an operational panel against this system). If the EXEC goes into a loop, you run the risk of occupying the link indefinitely and essentially rendering the connection defunct.

Keyword	Required/ Optional	Description
MSGLVLI|M	Optional	Specifies the informational WTO messages to be suppressed. The default issues all WTO messages. To override the default, code: `MSGLVLI(n)` to suppress all informational WTORs.
VTS	Optional	Causes IMFSUBEX to suppress all messages.
Note: If abbreviations for the keywords are used, they must be separated by blank spaces or commas.		

Determining Return Codes from IMFSUBEX

A return code from IMFSUBEX indicates whether an EXEC was submitted for processing to the requested BBI-SS PAS. It is provided as the step completion code for a batch invocation, &LASTCC for TSO invocation, RC for a REXX EXEC, and returned in R15 when invoked as a called subroutine. Possible return codes are:

Codes	Description of Condition Code
00	EXEC was submitted to the BBI-SS PAS
08	The requested BBI-SS PAS not available, or not at required service level
12	Either BBI or the site security exit denied request
16	Error in the parm string
20	Severe error (program abend)

In IMFSUBEX, to distinguish between the return code generated by the EXEC and the return code generated by IMFSUBEX, a value of 2048 is added to the return code from the EXEC. Therefore, if the return code you receive from IMFSUBEX is equal to or greater than 2048, then the EXEC has been successfully executed and ended.

For example, IMFSUBEX can call an EXEC where the calling EXEC has WAIT(YES) specified. This means the calling EXEC halts execution until the called EXEC completes before it completes (also known as synchronous execution). If IMFSUBEX calls such an EXEC and the EXEC passes a return code of 4 when it completes, then the overall return code that appears in the job log for the batch job would be 2052.

Note: For the called EXEC to set a return code, the EXEC must use an IMFEXEC EXIT statement to end the EXEC.

In another scenario, an AutoOPERATOR EXEC (for example, called EXEC1) or a TSO CLIST running in an TSO address space can call the IMFSUBEX subroutine which will schedule a second EXEC (for example, called EXEC2). If EXEC2 sets a return code of 4, then the &LASTCC variable would contain a value of 2052. You can use the IMFEXEC EXIT statement in an EXEC to set the return code.

The following example shows what happens when processing return codes using the WAIT parameter.

```
ENAME=SMFDUMP
 .
 .
"CALL 'BBI.BBLINK(IMFSUBEX)' 'SS(SSA1) EXEC("ENAME") WAIT(YES)'"
 IF RC LT 2048 THEN
  SAY 'EXEC' ENAME 'NOT SCHEDULED RC='RC
 ELSE
  SAY 'EXEC' ENAME 'SUCCESSFULLY SCHEDULED RC='RC-2048
 .
 .
```

If an IMFSUBEX is invoked to schedule an EXEC and the EXEC is not found in the SYSPROC data set, the return code is 8. If the EXEC is found in the SYSPROC data set and it is scheduled, upon termination, IMFSUBEX adds the value of 2048 to the return code set in the EXEC that terminated.

Submission from a Job Step

To submit an EXEC from a job step:

```
//stepname EXEC PGM=IMFSUBEX,PARM='parm-string'
//STEPLIB    DD DSN=BBI.BBLINK,DISP=SHR
```

To submit an EXEC STOPCICS that stops CICS:

```
//S1    EXEC PGM=IMFSUBEX,
//         PARM='SS(SSA1) EXEC(STOPCICS NOW)'
```

To pass different parameters to the EXEC depending on a previous job step's condition code:

```
//BACKUPD JOB (acct info),'BACKUP PROD-DB',other job parms
//*
//*        USER PROGRAM BACKS UP THE APPLICATION DATABASES
//*
//STEP1 EXEC PGM=userprog
//dd1     DD . . .
//dd2     DD . . .
//dd3     DD . . .
//*
//*        BACKUP OK>  RESTART DATABASES IN ONLINE SYSTEM
//*
//STEP2 EXEC PGM=IMFSUBEX,COND=(0,NE,STEP1),
//         PARM='SS(SSA1) EXEC(BACKUPDB 0 OK)'
//*
//*        ERROR IN THE BACKUP BUT MOST WORK COMPLETED.
//*        ATTEMPT TO RESTART DATABASES IN ONLINE SYSTEM,
//*        SEND MESSAGE TO WARNING SCREEN
//*
//STEP3 EXEC PGM=IMFSUBEX,COND=(8,NE,STEP1),
//         PARM='SS(SSA1) EXEC(BACKUPDB 8 ERROR)'
//*
//*        BACKUP ABENDED>  IF DAYTIME, SEND MESSAGE TO
//*        APPLICATION PROGRAMER WITH CICS SEND. IF
//*        NOT, SEND MESSAGE TO WARNING SCREEN.
//*
//STEP4 EXEC PGM=IMFSUBEX,COND=ONLY,
//         PARM='SS(SSA1) EXEC(BACKUPDB ABEND FAILED)'
```

Submission from a TSO Session

From a TSO session, there are two ways to invoke IMFSUBEX:

- With the CALL command:

  ```
  CALL 'BBI.BBLINK(IMFSUBEX)' 'parm-string'
  ```

- As a TSO command:

  ```
  IMFSUBEX parm-string
  ```

For example, to start the PAYROLL application from a TSO CLIST, you can use either a CALL command:

```
CALL 'BBI.BBLINK(IMFSUBEX)' 'SS(SSA1) EXEC(PAYROLL START)'
```

or a TSO command:

```
IMFSUBEX SS(SSA1) EXEC(PAYROLL START)
```

Submission from within Another Program

IMFSUBEX can be called from within another program. IMFSUBEX need not be authorized for this by the MVS Authorized Program Facility. The AutoOPERATOR BBLINK library must be in the STEPLIB concatenation. It should be the last library to avoid any negative impact on performance.

The first example is an Assembler Language example; the second is a COBOL.

```
                .
                .
       MAIN010  DS    0H
                LINK  EP=IMFSUBEX,PARAM=(STRING)
                .
                .
       STRING   DC    C'SS(*) EXEC(SSTATUS)',x'00'
```

With COBOL, a dynamic call is required.

```
                .
                .
   DATA DIVISION.
    01 PARM-STRING.
        05 PARM-DATA    PIC  X(30)
                       VALUE  'SS(*) EXEC(STOPDB IIDB002A)'.
        05 PARM-END     PIC  X(1)  VALUE LOW-VALUE.
    01 IMFSUBEX         PIC  X(8)  VALUE 'IMFSUBEX'
                .
                .
   PROCEDURE DIVISION.
                .
                .
       CALL  IMFSUBEX USING PARM-STRING.
```

Note: The parameter string can have up to a total of 256 bytes with the last byte being hex '00'.

Testing EXECs

You can use the information from any of the following sections and test your EXEC before you implement it as a BBPROC member. AutoOPERATOR also offers a full testing facility for you to test EXECs. See Chapter 13, “Testing and Debugging EXECs Interactively” on page 411.

However, you can also invoke your EXECs and minimize the effect they might have by employing the techniques in the following sections.

The techniques provide you with a few ways to examine your EXECs:

You can execute an EXEC...	**See...**
And not issue certain IMFEXEC statements, thereby minimizing impact of certain EXECs to your system	“Testing EXECs with IMFEXEC CNTL NOCMD Statements” on page 100
And examine variable substitution in the BBI-SS PAS Journal to see if variables are resolving correctly	“Testing EXEC with REXX Statement TRACE R” on page 101 and “Testing EXECs with SHARED Variables” on page 102
And not issue any WTOs you might have included	“Testing EXECs without Issuing WTOs” on page 103

Testing EXECs with IMFEXEC CNTL NOCMD Statements

By including the IMFEXEC statement IMFEXEC CNTL NOCMD, you can write an EXEC and run the EXEC on your system without actually executing the actions specified with the following IMFEXEC statements:

- IMFEXEC CMD
- IMFEXEC CICSTRAN
- IMFEXEC IMSTRAN
- IMFEXEC SUBMIT
- IMFEXEC RES EXIT
- IMFEXEC RES MCMD
- IMFEXEC RES VMCMD

Refer to “Using the IMFEXEC Statements” on page 237 for more information about these individual statements.

For example, you might use the IMFEXEC CMD to issue an MVS command, such as activate a VTAM terminal, in an EXEC. You can execute the EXEC and choose not to issue the MVS command by including the IMFEXEC statement:

```
"IMFEXEC CNTL NOCMD"
```

in the EXEC, prior to the IMFEXEC CMD statement.

You can track the results of the EXEC by examining the BBI-SS PAS Journal which indicates that the MVS command was not executed because of the IMFEXEC CNTL NOCMD statement.

Example

The following is a short example of how you might use IMFEXEC CNTL NOCMD.

```
"IMFEXEC CNTL NOCMD"
"IMFEXEC CMD #V NET,ACT,ID=BB010A"
```

Figure 14. Example of Using IMFEXEC CNTL NOCMD

The MVS command to vary VTAM terminal BB010A will not be executed when this EXEC is invoked. In the BBI-SS PAS Journal, you will see a message that looks like:

```
EM1101I FOLLOWING COMMAND BYPASSED DUE TO TEST MODE:
  IMFEXEC CMD #V NET,ACT,ID=BB010A
```

Figure 15. Example 1 of BBI-SS PAS Journal Entry

Testing EXEC-initiated EXECs with IMFEXEC CNTL NOCMD GLOBAL statements

You can test an EXEC-initiated EXEC and not execute the following IMFEXEC statements by using the IMFEXEC CNTL NOCMD statement with the parameter GLOBAL

- IMFEXEC CMD
- IMFEXEC CICSTRAN
- IMFEXEC IMSTRAN
- IMFEXEC SUBMIT
- IMFEXEC RES EXIT
- IMFEXEC RES MCMD
- IMFEXEC RES VMCMD

The statement:

```
"IMFEXEC CNTL NOCMD GLOBAL"
```

will prevent these statements from being executed in the calling and in the called EXEC of an EXEC-initiated EXEC.

Testing EXEC with REXX Statement TRACE R

By using the REXX statement TRACE R in your EXEC, you can see all the statements in the EXEC written to the BBI-SS PAS Journal and all the TSO variables resolved as the EXEC executes. This is useful if you want to insure that your TSO variables are being resolved as you expected. For a complete discussion for using TRACE R to debug your REXX EXECs, refer to *TSO Extensions Version 2: REXX/MVS User's Guide*.

Enter the statement:

```
TRACE R
```

at the line of the EXEC where you want to begin this test.

For example, if you were to invoke an EXEC called CALLRSTX and pass two parameters to it, type:

```
%CALLRSTX USER1 DETAIL
```

at any Command line.

The following is an example of the substitution that is logged to the Journal:

```
14:59:24  EM0025I  FOLLOWING MSG ISSUED FOR EXEC .. CALLRSTX ..
14:59:24       3 *-* ARG NAME PARM1 DETAIL GARBAGE
14:59:24         >>>   "CALLRSTX"
14:59:24         >>>   "USER1"
14:59:24         >>>   "DETAIL"
14:59:24       5 *-* /* DISPLAY THE INPUT PARAMETERS */
14:59:24       6 *-* IMFEXEC MSG 'PARM1 =' PARM1
14:59:24         >>>   "IMFEXEC MSG PARM1 = USER1"
14:59:24  PARM1 = USER1
14:59:25       7 *-* IMFEXEC MSG 'DETAIL =' DETAIL
14:59:25         >>>   "IMFEXEC MSG DETAIL = DETAIL"
14:59:25  DETAIL = DETAIL
```

Figure 16. Example 2 of BBI-SS PAS Journal Entry

In this BBI-SS PAS Journal entry, you can see the substitution for the ARG statement where the values you used to invoke the EXEC are passed to the ARG statement at line 3). Line 5 shows a comment from the REXX EXEC and lines 6 and 7 show the actual substitution of the variables.

Testing EXECs with SHARED Variables

Another technique you might use is to use the IMFEXEC VPUT statement to put variables into the SHARED variable pool instead of the LOCAL variable pool. For example, instead of using this statement:

```
"IMFEXEC VPUT (WORD1 WORD2  WORD4) LOCAL"
```

you can use the following statement:

```
"IMFEXEC VPUT (WORD1 WORD2  WORD4) SHARED"
```

By placing the variables WORD1, WORD2, and WORD3 to the SHARED pool, you can verify the values that were substituted. Use the command:

```
.D V SHARED
```

to see how the variables were resolved in the SHARED pool. Once you have verified them, you can then adjust your EXEC to put the variables back to the LOCAL pool.

Testing EXECs without Issuing WTOs

If you are writing EXECs with the IMFEXEC WTO statement and you want to run your EXEC without actually issuing the WTO, replace the IMFEXEC WTO statement with IMFEXEC MSG and the message will be written to the BBI-SS PAS Journal.

For example, if you have the following statement:

```
"IMFEXEC WTO 'THE WORLD IS COMING TO AN END' DESC(2)"
```

you can comment it out with comment marks (/*, */) and use:

```
"IMFEXEC MSG 'THE WORLD IS COMING TO AN END'"
```

This message would be written to the BBI-SS PAS Journal.

REXX EXEC Considerations

If you have the IBM REXX Compiler installed at your site, AutoOPERATOR supports tokenized REXX EXECs with the following considerations:

- The EXEC Management application does **not** display documentation (`DOC`) fields for tokenized REXX EXECs in its display fields.

  All comments are removed from the REXX EXEC by the compiler.

- The tokenized REXX EXECs must be stored in a SYSPROC library concatenation.

AutoOPERATOR does not support compiled REXX EXECs.

You can expect significant performance gains when you use tokenized REXX EXECs over interpreted EXECs. These gains, however, depend on the number of external calls (such as IMFEXEC commands) or subroutines used.

Wherever possible, REXX functions and subroutines should be built into the parent REXX EXEC. This is much more efficient because it eliminates the function or subroutine load time.

Once a REXX EXEC has been analyzed for performance and optimized, subroutines called many times using IMFEXEC SELECT EXEC can be copied internally to the parent and called using REXX CALL.

BBSAMP member AOXCPUST contains the tokenized version of AOXCPUSI.

Minimizing EXEC Processing Time

In general, BMC Software recommends you use Rules to perform basic automation tasks whenever possible. Rules are less prone to have errors and use less CPU than EXECs. The use of EXECs should be considered only after you have determined that the automation task cannot be accomplished with a Rule.

For AutoOPERATOR to perform automation efficiently with EXECs, the subsystem must be tuned to process EXECs as quickly as possible. The desirable level of throughput (or the number of EXECs processed per minute) for each site varies, depending on your automation requirements and the design of the EXECs.

There are some things you can do to ensure EXECs run more efficiently:

- Fix the dispatching priority of the subsystem.

  The subsystem (SS) must be run at a **fixed** dispatching priority. The priority of the SS must be higher than (or equal to) the regions that AutoOPERATOR is managing (for example: CICS, IMS, JES2). This ensures AutoOPERATOR can quickly respond to events in these regions.

- Allocate the correct number of EXEC threads.

  Adjust the number of EXEC threads (with the MaxNorm and MaxHigh parameters in BBPARM member AAOEXP00) to the **minimum** number required to achieve the level of throughput you want.

- Use the OS/390 Virtual Lookaside Facility (VLF) service which is available with OS/390 (MVS Version 3 and later).

  Using VLF allows AutoOPERATOR to perform EXEC processing with a minimum of I/O activity, and reduced I/O activity leads to less system overhead and improved performance.

  Refer to the IBM publication OS/390 *Initialization and Tuning Reference* for information about the VLF service. Refer to “Using VLF to Improve Performance” on page 105 for more information about AutoOPERATOR EXECs and VLF.

Using VLF to Improve Performance

This section contains information about AutoOPERATOR and VLF.

Implementing VLF

BMC Software recommends that you store your EXECs in VLF using the IKJEXEC VLF class. For more information about the IKJEXEC class, refer to the IBM publication *TSO Extensions Version 2, Customization*. This manual also contains some information about implementation considerations that you should review.

Because there are some known problems with running VLF and TSO, you must make sure all recommended IBM PTFs are applied.

Important Note
Loading your EXECs from VLF is transparent to the EXEC Management application. However, EXECs stored in VLF cache cannot be tested more than once per SS session by the AutoOPERATOR EXEC Testing Facility. The first time you issue the line command `T` to test the EXEC, the Testing Facility gets control of the EXEC with TSO OPEN SYSPROC and the test is run. However, subsequent attempts to test the EXEC cause the EXEC to be scheduled and the Testing Facility is **bypassed**. This occurs because once the EXEC is read into the VLF cache, the EXEC Testing Facility is not able to get control over the execution of the EXEC.

VLF and EXECs

Ordinarily when you execute an EXEC, for each EXEC, TSO will OPEN SYSPROC, read all the EXEC records into memory, and CLOSE SYSPROC. If the EXEC is present in VLF cache, then these three operations are eliminated, which means there is a considerable reduction in both CPU and I/O (and less DASD device and channel contention) when EXECs are in the VLF cache.

This is because VLF caches individual SYSPROC data sets. You must determine the appropriate amount of virtual storage to devote to the cache for this VLF class, which is specified with the MAXVIRT parameter. The MAXVIRT parameter is documented in the IBM publication *MVS/ESA Initialization and Tuning Reference*.

Note that if the specified cache is too small and too many EXECs are cached, thrashing in the cache can occur and performance could actually be worse than when EXECs are read directly from DASD. One possible remedy is for you to move the EXECs that are used more frequently into a smaller data set, place this data set first in the SYSPROC concatenation, and have VLF cache this data set.

Using the SYSEXEC DD: If the SYSEXEC DD is present, TSO will search it first for each EXEC, and **VLF has no effect on SYSEXEC**. Therefore, BMC Software recommends you do not use the SYSEXEC DD.

Restrictions: Note carefully the restrictions and considerations for updating VLF cached libraries, both on single and multiple MVS images. For more information, refer to the IBM publication *TSO Extensions Version 2, Command Reference* for documentation for the TSO VLFNOTE command.

Chapter 7. Accessing DB2 from AutoOPERATOR

This chapter describes how you can access DB2 from AutoOPERATOR with REXX EXECs if you have the BMC Software product RxD2/LINK product installed.

Access DB2 from REXX EXECs with RxD2/LINK

If RxD2/LINK is installed in the BBI-SS PAS, AutoOPERATOR REXX EXECs can issue dynamic SQLs to access and manipulate DB2 data. The REXX EXEC can ADDRESS DB2 as it can ADDRESS MVS.

This added facility allows:

- Accessing the DB2 catalog for information about DB2 objects (such as tables and plans)
- Accessing other DB2 tables to read external data that can govern AutoOPERATOR procedures
- Storing data collected by the EXECs for later queries and reporting using the full function of SQL

Refer to the *RxD2/LINK User Guide and Reference* for more information about customization and usage.

Note: The BBI-SS PAS requires authorization for the DB2 functions to be performed.

RxD2/LINK Common Functions for REXX EXECs

Several EXECs are delivered with RxD2/LINK to provide commonly used functions and reduce user coding. They are ready to use and can be invoked from any other EXEC.

Table 6. Common Function EXECs

Common Function EXECs	Description

RXBKLINE(mxlen,iline)	This EXEC truncates the character text in ILINE at a word boundary to a length no greater than MXLEN. It is useful in displaying a long SQL statement.
	
	If either argument is null, a null string is returned.
	
	`RXBKLINE(9,'This is an example') -> 'This is'`
	`RXBKLINE(9,'This too,is an example') -> 'This too,'`
	`RXBKLINE(72,'This is an example') -> 'This is an example'`

RXQCHAR(wname,wdata)	This EXEC builds a predicate for the character-type column WNAME from the string entered as a qualifier in WDATA. It is used to generate SQL predicates from user input specifying a selection qualifier for a column of a table.
	
	`RXQCHAR('NAME','DSN')  -> "NAME = 'DSN'"`
	`RXQCHAR('NAME','DSN*')  -> "NAME LIKE 'DSN%'"`
	`RXQCHAR('NAME','D+N')  -> "NAME LIKE 'D_N'"`
	`RXQCHAR('NAME','NULL')  -> "NAME IS NULL'"`
	`RXQCHAR('NAME','^NULL')  -> "NAME IS NOT NULL"`
	`RXQCHAR('NAME',"^='DSN'")  -> "NAME ^= 'DSN'"`
	`RXQCHAR('NAME',"<'DSN'")  -> "NAME < 'DSN'"`
	`RXQCHAR('NAME',">'DSN'")  -> "NAME > 'DSN'"`

RXQNUM(wname,wdata)	This EXEC builds a predicate for the numeric-type column WNAME from the string entered as a qualifier in WDATA. It is used to generate SQL predicates from user input specifying a selection qualifier for a column of a table.
	
	`RXQNUM('NAME','123')  -> "NAME = 123'"`
	`RXQNUM('NAME','<123')  -> "NAME < 123'"`
	`RXQNUM('NAME','^=123')  -> "NAME ^= 123'"`

RXSAMPEX	This is a sample EXEC to process SQL statements or DB2 commands and display the results in line mode. It does not require ISPF and therefore is usable in any address space; for example, batch jobs, NetView, or AutoOPERATOR EXECs. **Note:** The RXSAMPEX EXEC is invoked by the two sample batch jobs, RXBATSQL and RXBATCMD, that are distributed as members in BBSAMP.

Table 6. Common Function EXECs (Continued)

Common Function EXECs	Description

RXSETSQL	This EXEC constructs an SQL statement from the text pointed to by a cursor in an ISPF/PDF edit panel.
	
	⋮
	
	`SQL = RXSETSQL()`
	
	⋮
	
	`A = WORDPOS('INTO',SQL)`

RXVODS(wdsn)	This EXEC verifies that the data set name specified in WDSN is valid. It checks that the data set exists and that the data set is either sequential or a PDS with a member name specified. The EXEC is used to verify an output data set before the data set is used.
	
	`WMSG = RXVODS($VLSTDS)`
	`IF WMSG = 'OK' THEN DO`
	`"ALLOC DD(LCOUT) DA("$VLSTDS") SHR REUSE"`
	`IF RC = 0 THEN NOP`
	`ELSE WMSG = 'ALLOC ERROR' RC`
	`END /* WMSG = OK THEN */`

RxD2/LINK Special Functions for REXX EXECs

Several special functions are provided with RxD2/LINK that are required or useful when accessing DB2.

In REXX, you invoke a function by issuing:

```
V1 = FUNC(ARG1,ARG2)
```

where V1 is the variable into which the function FUNC places the result.

Table 7. Special Functions

Special Function	Description

CONVSTCK(tod)	Converts the 8-byte TOD clock into display format of YYYYDDD HHMMSSTH. Valid from 1/1/1988 onward. The 8-byte TOD format is such that bit 51 equals 1 microsecond (see the IBM publication *370 Principles of Operations*).
	
	`TSTMP = 'A42AE3F94CE5BB31'X`
	`SAY "TIMESTAMP=" CONVSTCK(TSTMP)`
	
	DEFAULT None
	
	RETURN 'value' if function completes successfully
	NOGO 'reason' if function fails for the reason given

CTOD(tod)	Converts the 8-byte TOD clock time into display format of HHMMSSTH. The 8-byte TOD format is such that bit 51 equals 1 microsecond (see the IBM publication *370 Principles of Operations*).
	
	`CPUT = '0000000160B79C00'X`
	`SAY "CPUT=" CTOD(CPUT)`
	
	DEFAULT None
	
	RETURN 'value' if function completes successfully
	NOGO 'reason' if function fails for the reason given

F2C(f)	Do a floating point conversion on variable f and return the floating point number in display format.
	
	`/* TEST F2C */`
	`A = '4498765432100000'X`
	`SAY "F2C=" F2C(A)`
	
	DEFAULT None
	
	RETURN 'value' if function completes successfully
	NOGO if function fails

Table 7. Special Functions (Continued)

Special Function	Description

GBLVAR
(GETV,varname)
(SETV,varname)
(DROP,varname)
(UPDV,varname)
	Create and manage the global variable environment. The global variable environment is created at first use. Subsequent environment shares the same environment. The environment is destroyed at the EOT of the task that created the environment.
	
	GETV
	Gets the global variable varname and places its content in the local variable varname.
	
	`SAY "TESTVAR=" TESTVAR`
	`SAY "GBLVAR('GETV','TESTVAR')=" GBLVAR('GETV','TESTAVR')`
	`SAY "TESTVAR=" TESTVAR`
	
	SETV
	Gets the local variable varname and creates a global variable varname. If the global variable varname already exists, it is not replaced.
	
	`TESTVAR= "TEST VARIABLE FOR TEST GBLVAR"`
	`SAY "GBLVAR('SETV','TESTVAR')=" GBLVAR('SETV','TESTVAR')`
	
	DROP
	Drops the global variable varname.
	
	UPDV
	Gets the local variable varname and updates the global variable varname. If the global variable varname does not exist, the function is treated like "SETV".
	
	DEFAULT None
	
	RETURN OK if function completes successfully
	OK 'warn' if function completes with a warning
	NOGO 'reason' if function fails for the reason given

P2C(p)	Do an unpack on variable p and return the packed decimal number in display format.
	
	`/* TEST P2C */`
	`A = '123456789C'X`
	`SAY "P2C=" P2C(A)`
	
	DEFAULT None
	
	RETURN 'value' if function completes successfully
	NOGO if function fails

Table 7. Special Functions (Continued)

Special Function	Description

UENV(hcename,pgm)	Identify to REXX Host Command Environment (HCE) called hcename, such that pgm will receive control for ADDRESS hcename. The hcename currently is required to be DB2.
	
	`SK = UENV(DB2)`
	`IF SK ^= "OK" THEN DO`
	`  SAY "UNABLE TO ENABLE RXDB2"`
	`  EXIT 16`
	`  END`
	
	DEFAULT hcename = DB2
	pgm = RXDB2
	
	RETURN OK if function completes successfully
	NOGO if function fails

VARSPF(varname)	A compound variable (AA.1) cannot be used in an ISPF dialog. Function VARSPF(AA.1) creates a new simple variable AA1 containing the same data as AA.1 so it can be used in an ISPF dialog.
	
	The function first compresses out the period(s) in the compound variable name and then ensures that the resulting variable name is no more than 8 characters long.
	
	`IF DATATYPE(SQLEMSG.0) = NUM THEN`
	`  DO I = 1 TO SQLEMSG.0`
	`  SQLEM.I = SPACE(SQLEMSG.I)`
	`  A = VARSPF("SQLEM."I)`
	`  END /* I LOOP */`
	
	DEFAULT None
	
	RETURN OK if function completes successfully
	NOGO if function fails
	TRUNCATED if function has to truncate the variable name

WAITSEC(n)	Wait n seconds before continuing to process.
	
	`DO I = 1 TO 5`
	`  A = WAITSEC(2) /* WAIT 2 SECONDS */`
	`  SAY "LOOP COUNT=" I "TIME=" TIME()`
	`  END`
	
	DEFAULT n = 5 (seconds)
	
	RETURN OK if function completes successfully
	NOGO if function fails

Chapter 8. Interacting with VTAM-Applications with OSPI

This chapter describes the AutoOPERATOR Open Systems Procedural Interface (OSPI) feature and how to use to interact with VTAM-based applications and for automation tasks.

Overview

AutoOPERATOR provides the Open Systems Procedural Interface (OSPI) as an interface to VTAM-based products. OSPI provides a means for REXX- or CLIST-based automation procedures to interface with any LU2 (3270) VTAM application that uses full screens to communicate with users.

With OSPI, AutoOPERATOR has logon capabilities and complete access to any VTAM application's data stream. In this way, AutoOPERATOR can interact with the application by analyzing the output data and issuing the VTAM application's own commands.

By automatically interfacing with critical VTAM applications and simulating a user at a VTAM terminal, OSPI can communicate with various data center software products and decrease the number of physical terminals required.

OSPI includes three components:

- IMFEXEC commands that allow EXECs to communicate with VTAM applications
- A Scripting application that automatically generates IMFEXEC command statements by recording your interactions with a terminal
- A Debugging facility

These components are described in the following sections:

- “OSPI Sessions” on page 115 provides a general overview about how the Scripting application generates OSPI EXECs.
- “OSPI Scripting Application” on page 116 provides detailed information about using the Scripting application.
- “Application Termination” on page 123 describes how to customize EXECs generated by the Scripting application.
- “OSPI Debugging Facilities” on page 126 describes the Debugging facility.

When to Use OSPI

AutoOPERATOR communicates with MVS and its subsystems using standard software interfaces; for example, the Subsystem Interface (SSI) is used to communicate with MVS. However, many VTAM applications do not provide a software interface but require use of a 3270 terminal instead.

The OSPI facility provides access from an AutoOPERATOR EXEC to these VTAM applications without requiring a physical 3270 terminal. OSPI allows *most 3270-operator actions to be emulated by an EXEC.* Use this facility when you need to access VTAM applications that ordinarily require an operator to actually log on to a 3270 terminal.

How to Use OSPI

The first step in automating a function using OSPI is to use the Scripting application to record the appropriate interactions with a VTAM application in an EXEC. The generated EXEC contains only OSPI IMFEXEC commands. It will not contain any conditional logic or other commands. Refer to “Using the IMFEXEC Statements” on page 237 for descriptions of the OSPI IMFEXEC command statements.

Depending upon the function being implemented, you may need to further customize the generated EXEC by combining the appropriate logic and commands with the OSPI IMFEXEC commands.

After the EXEC has been customized, it is ready to be executed. As with any EXEC, it should be thoroughly tested before it is installed into your production system.

Customization Required to Use OSPI

A session between OSPI and a VTAM application requires that OSPI function as a 3270 terminal. For OSPI to do this, some OSPI virtual terminals must be defined to VTAM. In addition, some applications, such as CICS and IMS, might require local definitions for the OSPI terminals. Finally, your site must be running a release of VTAM of V3 or higher.

These definitions must be implemented and activated prior to using the Scripting application or executing an OSPI EXEC. See the *MAINVIEW AutoOPERATOR Customization Guide* for more information about VTAM and application definitions required for OSPI virtual terminals.

OSPI Sessions

There are two types of OSPI sessions: scripting sessions and EXEC sessions. You can use both types of sessions when using OSPI to automate a function. Normally, scripting sessions are used first to record the appropriate interactions with VTAM applications in EXECs. Then, EXEC sessions are used when the functions are automatically performed by AutoOPERATOR.

All OSPI sessions follow the same basic flow regardless of the session type, the application they are interfacing with, or the task they are performing:

1. A session is established between an OSPI virtual terminal and a VTAM application.
2. Data is then exchanged between the virtual terminal and the application.
3. The session is terminated.

All of these tasks are accomplished using either the Scripting application or IMFEXEC command statements in an EXEC.

Establishing a Session

To initiate a scripting session, specify the parameters on the OSPI Script Development panel and press ENTER. The Scripting application uses these parameters to establish the scripting session and to generate a corresponding IMFEXEC LOGON command. This enables the generated EXEC to log on to the same VTAM application with the same parameters used in the scripting session.

Exchanging Data

After a successful logon, the Scripting application automatically receives the first buffer from the VTAM application. The first panel output by the VTAM application is displayed under the TS.

No additional IMFEXEC commands are generated at this point because the previously generated IMFEXEC LOGON command automatically receives the output sent by the application. When the generated EXEC is executed, control is not returned to the EXEC until the first complete buffer image is received and available for processing by the EXEC.

As you interact with the application by sending and receiving new data, the Scripting application records these actions using IMFEXEC TYPE and IMFEXEC TRANSMIT commands. This enables the generated EXEC to automatically perform the same functions as a real terminal user might.

Terminating a Session

When you terminate the session with the VTAM application, the OSPI Session Termination panel is displayed. You then have the option of saving or cancelling the script. If the script is saved, an IMFEXEC LOGOFF command is generated, the EXEC is saved in the first data set of your SYSPROC concatenation, and the command `.RESET BLDL SYSPROC` is automatically done.

OSPI Scripting Application

User interaction with OSPI is simplified with the Scripting application. The Scripting application records your keystrokes as you make them, and you can use this Scripting application to create complex procedures to drive 3270 applications without writing a line of procedural code.

The OSPI Scripting application can create procedures in either CLIST or REXX.

Accessing the OSPI Scripting Application

Access the Scripting application by selecting option 7, OSPI, from the PRIMARY OPTION MENU. The OSPI Script Development panel, shown in Figure 17 on page 117, is displayed.

The following topics provide details about accessing a VTAM application using the OSPI Scripting application.

To learn about...	**See...**
Specifying the appropriate data for establishing a session	“OSPI Script Development Panel” on page 117
Differences you may see when accessing an application under the Scripting application versus directly through VTAM	“Interacting with the Application” on page 119
Making data in the terminal buffer available to a generated EXEC	“Retrieving Screen Data into Variables” on page 122
The options available for ending an OSPI session with an application	“Application Termination” on page 123

OSPI Script Development Panel

```
BMC SOFTWARE ------------------ OSPI Script Development --------- AutoOPERATOR
COMMAND  ===>                                                      TGT ===> SYSB
                                                                DATE ---  01/01/15
                                                                TIME ---  13:15:51

   To begin a Scripting Session, specify the following and press ENTER


   Member name ===> OSPI               Application for LOGON ===>

   Overwrite existing member ===> Y                  Hot key ===> PF 11 (01-12)

              Logmode to use ===> D6327802             Debug ===> N

   User data   ===>

   ACB to use  ===>                       Language Option ===> CLIST (REXX/CLIST)

   Process initial receive ===> Y

Press END to abort request
```

Figure 17. OSPI Script Development Panel

The OSPI Script Development Panel is used to specify the parameters, such as the application to be accessed and the terminal type to be emulated, for establishing the scripting session. Figure 17 shows default values in all fields.

Following is a description of each field:

Member name
Name to be used when the generated EXEC is stored in the SYSPROC data set.

Note: If multiple data sets are concatenated to the SYSPROC DD, the member is stored in the first data set in the concatenation.

Application for LOGON
Name of the application (as specified in a VTAM APPL statement) you want to establish a session with. VTAM interpret tables are not used so this name may differ from the name you enter when logging on at a terminal.

Overwrite existing member
If the member named in the `Member name` field already exists, verify that you want to overwrite it.

Hot key
ISPF may process certain program function (PF) keys, such as SPLIT and SWAP, before passing them to OSPI. For this reason, you must use the OSPI hot key in place of any PF or PA keys. The default hot key is PF11. You can reassign it to any non-ISPF specific PF key.

When you press the hot key, a hot key pad is displayed to allow you to specify which PF/PA keystroke should be passed to the application.

See “Program Function Keys” on page 120 for information about using the hot key pad.

Logmode to use

The logmode associates certain terminal characteristics, such as support for extended attributes (color, reverse video, and so on) and screen size, with the OSPI terminal emulation. The type of terminal that is emulated may affect the application displays seen by the scripting user and the data available to the generated EXEC. See “Extended Attributes” on page 119 for more information about selecting an appropriate logmode.

The logmode must be a valid VTAM MODEENT in the MODETAB associated with the OSPI ACB (specified in the `ACB to USE` field). The default is a 3278 Model 2, specified as `D6327802`. This is the recommended logmode to use.

Debug

Specifies whether or not debugging information will be written to the BBI-SS PAS Journal log and to the OSPISNAP data set.

User data

Text (such as userid) to be passed to the application during session establishment.

ACB to use

ACB to be used for the OSPI virtual terminal. If you do not specify an ACB, an ACB is selected from the OSPI ACB pool. The ACB generated on this panel is intentionally not carried forward into the generated EXEC.

See the *MAINVIEW AutoOPERATOR Customization Guide* for more information about ACB definitions required for OSPI virtual terminals.

Language Option

Specifies the CLIST language to be used in the generated EXEC where:

– Specifying REXX causes a REXX EXEC to be generated.

– Specifying CLIST causes a TSO CLIST to be generated.

Process Initial Receive

Indicates whether or not OSPI should attempt to receive an initial panel (buffer) before allowing data to be entered and sent to the application. The default is Y and results in OSPI waiting for the first panel (buffer) to be received from the application before allowing the terminal operator to enter data.

In most cases, the default should be used. However, when logging on to an application that does not display an initial panel before allowing the terminal user to enter data (for example, a CICS system without a "Good Morning" transaction), you must specify `N` to avoid an unending wait. Refer to “Receive Complete Detection” on page 121 for more information.

After you press ENTER, OSPI attempts to establish a session using the parameters specified. If a session is successfully established, the first panel output by the VTAM application is displayed under the TS. You can now interact with the application to perform and record the function you want to automate with an OSPI EXEC.

If a session cannot be established, the OSPI Session Termination panel is displayed. See “OSPI Session Termination Panel” on page 127 for information about interpreting the VTAM error codes displayed on the panel.

Interacting with the Application

Most of the time, accessing an application under the OSPI Scripting application is identical to accessing the same application directly through VTAM. However, there are some differences in the following areas:

- 3270 attributes, such as extended color or extended highlighting
- Program function (PF) keys
- ISPF jump function
- Screen size and usage
- Receive complete detection

3270 Attributes

Displays that contain extended color or extended highlighting attributes may look slightly different when executing under OSPI because OSPI does not honor these attributes. Data streams containing these attributes are not properly interpreted and may cause errors, such as treating fields with extended attributes as protected.

These attributes are not honored because extended attributes do not occupy a position in the screen buffer and, therefore, OSPI EXECs cannot benefit from their settings.

Extended Attributes: OSPI terminal emulation uses the characteristics associated with the terminal LOGMODE specified on the OSPI Script Development panel and, ultimately, on the generated IMFEXEC LOGON command. Therefore, always choose a LOGMODE that designates the least amount of terminal capabilities possible.

Some applications use reverse video to create bars on a screen whenever the LOGMODE indicates that the terminal supports extended attributes. OSPI EXECs would simply see blanks in the field that contained the reverse video bars. However, the same application may use character data instead of the reverse video bars when the LOGMODE indicates that the terminal does not support extended attributes.

Program Function Keys

The OSPI Scripting application may execute under ISPF, and thus ISPF may process certain program function (PF) keys, such as SPLIT and SWAP, before passing them to OSPI. For this reason, it is necessary to use the OSPI hot key in place of any PF or PA keys. The default hot key is PF11. It may be reassigned to any non-ISPF specific PF key.

When you press the hot key, the OSPI Transmission Keystroke panel, shown in Figure 18, is displayed. Enter the option number associated with the PF or PA key you want to transmit to the application.

```
BMC Software  ------------- OSPI transmission keystroke ---------- AutoOPERATOR
COMMAND ===>

   Please select action from list below:

   Keystrokes:
   PF1  -  1           PF13 - 13                  ENTER - 25
   PF2  -  2           PF14 - 14                  CLEAR - 26
   PF3  -  3           PF15 - 15
   PF4  -  4           PF16 - 16                  PA1   - 27
   PF5  -  5           PF17 - 17                  PA2   - 28
   PF6  -  6           PF18 - 18                  PA3   - 29
   PF7  -  7           PF19 - 19
   PF8  -  8           PF20 - 20          Other Options:
   PF9  -  9           PF21 - 21
   PF10 - 10           PF22 - 22         Cancel Session - 30
   PF11 - 11           PF23 - 23          Read variable - 31
   PF12 - 12           PF24 - 24          Attempt read  - 32

              Variable name ===>
         SHARED variable name ===>
              Select Option ===>
```

Figure 18. OSPI Transmission Keystroke Panel

ISPF Jump Function

The OSPI Scripting application is designed to execute under ISPF. Therefore, you must be careful when entering an equal sign (=) into any application screen OSPI displays.

When an equal sign is entered under ISPF, ISPF passes PF3 to the application as an indication that the application should terminate. OSPI does not process the PF3 but instead passes it to the application being scripted. ISPF continues passing PF3 until OSPI terminates. OSPI will continue passing PF3 to the scripted application and will never terminate. Therefore, attempting to enter the ISPF equal sign under the Scripting application **may** cause the TS to loop.

Screen Size and Usage

The OSPI Scripting application is designed to execute under ISPF. An application, such as OSPI, executing under ISPF must define the attributes associated with its display to avoid having a random attribute value assigned. Since ISPF does not support extended attributes, a position on the screen is always required to specify an attribute. The OSPI Scripting application uses line one, column one to specify an attribute and avoid having some random assignment.

With the Scripting application, you cannot enter data in line one, column one. Data is also not displayed in line one, column one. The first 80 bytes of output are shifted one byte to the right and byte 80 is not displayed if the buffer image to be displayed does not start out with an attribute byte. The date is shifted back one byte to the left before transmission to the application.

This restriction does not apply to OSPI EXECs. If necessary, EXECs generated by the Scripting application can be manually edited to specify that data be entered in row one, column one.

In addition, OSPI can only support screen sizes between 24 lines and 43 lines with a column width of 80. Any other screen size specified in a LOGMODE used by an OSPI virtual terminal causes errors.

Receive Complete Detection

When you send a new screen of data to the application, the Scripting application automatically tries to receive new data from the application. OSPI does not unlock the keyboard until the application has finished sending data.

Depending on the protocol used by an application, either the Change Direction Indicator (CDI) or End Bracket (EB) is used to determine when the application is finished sending data. If an application sends one of these indicators prematurely, it may be necessary to explicitly request that an additional receive be issued to receive data sent after the erroneous CDI or EB.

The OSPI Transmission Keystroke panel is used to request that an additional receive be issued. To do this:

1. Use the OSPI hot key to access the keystroke panel.
2. Select option 32, ATTEMPT READ, when the panel is displayed.

   If ATTEMPT READ is not issued, the data sent after the CDI/EB is not retrieved until after the next transmission of data from OSPI to the application.

Each time an ATTEMPT READ is issued, an IMFEXEC RECEIVE command is generated. IMFEXEC RECEIVE is not normally needed because OSPI automatically receives new data after IMFEXEC TRANSMIT.

Examples of applications that may require this special processing are Netview and VM/CMS.

Retrieving Screen Data into Variables

In addition to the IMFEXEC commands necessary to communicate with VTAM applications, the Scripting application can also generate the IMFEXEC SCAN commands necessary to retrieve data from the screen buffer into a variable.

You can use the OSPI Transmission Keystroke panel to request that some specific data in the screen buffer be read into a variable. This panel is accessed using the PF key designated as the OSPI hot key (default is PF11).

When the OSPI Transmission Keystroke panel is displayed, enter 31 (the `Read Variable` option) in the `Select Option` field of the panel. Also type in the name of the variable you want to create in the `Variable name` field. This causes data to be read into a variable. The default variable name is OSIVAR.

You can also specify a SHARED variable name in the `SHARED variable name` field which places the data into the SHARED variable using the given name.

When you press ENTER, the application screen that was displayed when you pressed the hot key is redisplayed. However, this display is used only to tell OSPI which data you want to retrieve from the screen. You cannot interact with the application at this point.

Position the cursor to the beginning of the data you want to read into a variable and press ENTER. Now position the cursor to the last position of the data you want to read and press ENTER. This sequence causes OSPI to generate an IMFEXEC SCAN command for the row, column, and length that was indicated by the cursor in the previously described sequence; for example:

```
IMFEXEC SCAN SESSION(&OSISESS) ROW(18) COL(6) LENGTH(6) +
  VAR(OSIVAR)
```

Of course, OSPI does not know how you want to use this data in your EXEC. You must edit the EXEC to make proper use of the data. However, it is much easier to retrieve the data using the `read variable` option than by calculating the correct row and column positions manually.

Application Termination

VTAM applications have a variety of methods by which you can request termination. For example, one application may terminate a session when `logoff` is received and another application may terminate a session when PF key 2 is received.

Since the data required to request session termination varies by application, OSPI does not know when such a request has been sent.

Each time the Scripting application sends data to a VTAM application, it also attempts to receive data. This read attempt usually fails after a request to terminate is sent. This failure causes OSPI to display an error panel. An example of this error panel is shown in Figure 19.

```
BMC Software ----------------- OSPI Session Termination -------- AutoOPERATOR
COMMAND ===>

    The OSPI session has terminated.

    Outstanding function was: RECEIVE DATA

    VTAM ACB error flag: 00

    Diagnostic information:

           RPL RTN/FDBK=0C0B, SENSE=00000000
           REQ CANCELLED DUE TO SESSION
           THE SESSION HAS BEEN TERMINATED

 Note: The above information may indicate that the session was terminated
       normally or abnormally.

 Press ENTER to display last buffer image, PF3  to save script and return.
 Enter CANCEL to skip script saving.
```

Figure 19. Example of Error Panel

The unsuccessful read looks like a true error to OSPI because it does not know that the last data sent requested application termination. If you receive this panel after you requested application termination, no true error occurred and you can ignore the error panel.

Note: You can also terminate a session using option 30, Cancel session, from the hot key pad. However, this is not recommended because the appropriate application clean-up may not be performed.

Customizing OSPI EXECs

The first step in automating a function using OSPI is to use the Scripting application to record the appropriate interactions with a VTAM application in an EXEC. The generated EXEC will contain only OSPI IMFEXEC commands. It will not contain any conditional logic or other commands.

Depending upon the function being implemented, you may need to further customize the generated EXEC by combining the appropriate logic and commands with the OSPI IMFEXEC commands. This section provides customization information in the following areas:

- "OSPI Control Variables" on page 124
- "Disconnect/Reconnect Feature" on page 125
- "Establishing Multiple Sessions" on page 125
- "Using Passwords in OSPI EXECs" on page 125

Note: This chapter discusses the OSPI IMFEXEC statements in general terms. See "Using the IMFEXEC Statements" on page 237 for information about specific parameters, return codes, and so on.

OSPI Control Variables

OSPI maintains a set of control variables that indicate the state of each OSPI session. These control variables are maintained in the EXEC's local variable pool. They are updated each time a new buffer image is received from the application.

The variables are:

OSISESS Session identifier. Must be used with the SESSION keyword on all OSPI IMFEXEC commands (except LOGON) to identify the session you are addressing.

OSIKSTAT Current keyboard status, either LOCKED or UNLOCKED.

OSIAPPL Name of the VTAM application associated with the OSPI session.

OSIROW Current cursor position, 1 to 43.

OSICOL Current cursor position, 1 to 80.

OSILNCNT Number of rows for the terminal type being emulated, 24 to 43.

OSILNnn Each OSILNnn represents one line of the current virtual screen buffer image. For example, OSILN2 contains line 2 of the current screen buffer image.

Before an EXEC can use one of the variables, it must be retrieved with an IMFEXEC VGET command; for example, IMFEXEC VGET OSISESS LOCAL.

Disconnect/Reconnect Feature

When an EXEC terminates, any OSPI sessions that have been established are either terminated or disconnected.

The IMFEXEC LOGOFF command without the DISCONNECT parameter results in a termination request being sent to the VTAM application. When the DISCONNECT parameter is specified, the VTAM session is not terminated and another EXEC may resume the session (RECONNECT) by issuing an IMFEXEC LOGON command with the SESSION parameter. The application will not be aware of any DISCONNECT/RECONNECT activity.

When an EXEC tries to reconnect a session, it is important that you check the condition code (IMFCC) after the IMFEXEC LOGON command. Reconnect sometimes fails due to applications terminating OSPI sessions when no activity occurs within a specified time. If IMFCC indicates that a reconnect is not successful, you must reestablish a new session.

To make use of the DISCONNECT/RECONNECT feature, the EXEC that initially establishes the session must store the session identifier (contained in the OSISESS variable) in a shared variable. This variable can then be retrieved by subsequent EXECs to reconnect. A unique shared variable name must be used for each different session that is concurrently maintained.

If an EXEC does not issue an IMFEXEC LOGOFF command, all sessions are automatically terminated. The result is the same as if explicit IMFEXEC LOGOFF commands had been issued for each session.

Establishing Multiple Sessions

An EXEC may establish sessions with multiple VTAM applications concurrently; however, a different OSPI control variable prefix must be used for each session. If different prefixes are not used for each session, the information for one session overlays the information for another session.

The default prefix for the OSPI control variables is OSI. The PREFIX keyword on the IMFEXEC LOGON command allows any three character prefix to be used for the variables.

Using Passwords in OSPI EXECs

Many of the applications that OSPI EXECs will access require passwords for logon. If the Scripting application is used to create the EXEC, the password is stored in the EXEC. For security reasons, BMC Software recommends that you edit the EXEC to replace the password literal with a variable.

One approach for handling this situation is to schedule an EXEC at AutoOPERATOR startup which requests the operator to enter the password. The password can then be stored in a global variable that can be retrieved by any OSPI EXEC needing access to the application.

OSPI Debugging Facilities

OSPI provides several facilities to aid in debugging EXECs and scripts.

Return Codes

Each of the IMFEXEC commands that interface with OSPI provides return code information in the IMFCC variable. Examining the value of IMFCC after issuing the IMFEXEC command can be useful during script development. During this phase, it may even be beneficial for you to record the IMFCC value in the BBI-SS PAS Journal log using the IMFEXEC MSG command.

After an EXEC has been fully debugged, IMFCC checks or messages that were added solely for debugging purposes should be removed. However, the IMFCC check for certain IMFEXEC commands should be retained even after development has been completed. For example, IMFCC after an IMFEXEC LOGON that specifies the SESSION parameter (reconnect) should always be retained.

Error Messages

Certain error conditions cause OSPI to generate error messages in the BBI-SS PAS Journal log. For example, error message OS5001E is produced if an attempt is made to enter data in a protected field. When a script is not functioning properly, it is always advisable for you to examine the BBI-SS PAS Journal log for error messages. If an error message is produced, additional information about the error can be found using the BBI Message application.

OSPI Control Variables

OSPI maintains a set of variables for each active or disconnected session. You may benefit from examining the value of one or more of these variables during EXEC development. See “OSPI Control Variables” on page 124 for more information about the control variables.

OSPISNAP

The OSPISNAP DD can be used to gather additional debugging information for OSPI EXECs and for the Scripting application. You must add the DD card to the BBI-SS PAS JCL and restart the BBI-SS PAS before directing any debugging information to it. The DCB characteristics for the OSPISNAP DD statement are: `RECFM=VBA, LRECL=125, BLKSIZE=1632`. The blocksize can be modified to fit your DASD requirements.

OSPISNAP may be routed to a SYSOUT class or to a data set. Two kinds of output can be directed to the OSPISNAP: session information and debugging information requested by BMC Software.

Session information is requested using the IMFEXEC SESSINF command. This command causes the following information to be recorded in the OSPISNAP data set:

- OSPI ACB associated with the current session
- Application that OSPI is in session with
- Keyboard status
- Cursor position
- Contents of screen buffer

Since the screen is not visible to a human developing an OSPI EXEC, you may find it helpful to use the IMFEXEC SESSINF command during the debugging phase.

If BMC Software support personnel request debugging information, you can obtain it by specifying the DEBUG keyword on the IMFEXEC LOGON command or on the Scripting panel. When you use the DEBUG keyword, the information is also written to the OSPISNAP data set when the EXEC is invoked. When DEBUG is turned on, additional messages are also written to the BBI-SS PAS Journal log.

Note: If you specify Y for the DEBUG option on the Scripting panel, debugging information is written to the OSPISNAP data set only during script development and not written to the OSPISNAP data set during EXEC execution.

OSPI Session Termination Panel

When a session between OSPI and a VTAM application is terminated, the OSPI Session Termination panel is displayed. This panel contains VTAM diagnostic information.

The following table contains error codes for some of the common reasons OSPI sessions are terminated.

ACB Error Flag	RPL Return and Feedback codes	Sense	Cause of Error
5A	N/A	N/A	OSPI terminal ACB cannot be opened
N/A	1012	087D0001	Application to log on to cannot be located
N/A	144B	00000000	OSPI terminal logmode cannot be located
N/A	0C0B	00000000	Application terminated the session
N/A	0006	00000000	Application terminated the session

When the diagnostic information indicates that the application has terminated the session, an error may not have actually occurred. See “Application Termination” on page 123 for more information about application termination and the OSPI Session Termination panel.

See “OSPI Script Development Panel” on page 117 for more information about specifying ACB, application, and logmode names.

Chapter 9. Performing Automation Using AOAnywhere

This chapter describes the AOAnywhere API and the syntax required to use it.

AOAnywhere is an application programming interface (API) that allows MAINVIEW AutoOPERATOR users to perform a variety of automation functions from outside the BBI-SS PAS address space. You can invoke AOAnywhere functions from

- The TSO/E command line
- Inside a REXX EXEC or TSO/E CLIST EXEC

You can invoke these functions to operate locally on a BBI-SS PAS running on the same system that the command is invoked from, or route functions to a remote system.

Overview

AOAnywhere allows AutoOPERATOR IMFEXEC automation functions to be invoked from address spaces outside of AutoOPERATOR using a new command: AOEXEC. The equivalent of the following IMFEXEC commands are available as AOEXEC commands:

- AOEXEC ALERT
- AOEXEC MSG
- AOEXEC NOTIFY
- AOEXEC SELECT
- AOEXEC SYSINFO
- AOEXEC VDEL
- AOEXEC VGET
- AOEXEC VLST
- AOEXEC VPUT
- AOEXEC VDELL
- AOEXEC VGETL
- AOEXEC VLSTL
- AOEXEC VPUTL

Sysplex Support

AOAnywhere functions can be invoked either locally (meaning on a BBI-SS PAS running on the same system that the command is invoked on) or remotely to a BBI-SS PAS.

To invoke a command on a remote system:

- A BBI-SS PAS must be active and available on the local system.
- XCF connectivity must exist between the local and the remote BBI-SS PAS.

Why Use AOAnywhere

AOAnywhere is a powerful function that allows access to many automation functions previously available only by using AutoOPERATOR IMFEXEC commands (in REXX EXECs or CLIST EXECs) **within the AutoOPERATOR subsystem**. AOAnywhere allows such access through an interface that operates outside of the subsystem. You can perform tasks that are part of production control or perform tasks that are part of a helpdesk system such as set and read variables or create or delete AutoOPERATOR ALERTS from REXX EXECs without going through the subsystem.

In previous releases of AutoOPERATOR (prior to version 6.1.00), you could use the IMFSUBEX interface to invoke EXECs but this method was slow and did not allow for two-way exchange of information. Only a return code issued by the invoked EXEC could be returned.

AOAnywhere functions are very fast; they allow sharing variable pools with invoked EXECs and access to a host of other functions. In most instances AOAnywhere offers a faster IMFSUBEX replacement while providing additional functionality.

Manual process intervention is also simpler. For example, when a helpdesk operator becomes aware of a network problem before automation does, the operator can generate an AutoOPERATOR ALERT and (with the MAINVIEW AutoOPERATOR Elan Workstation component installed) page additional personnel through an ISPF application. Logging on to the subsystem is not required and the operation itself can be executed in a few minutes.

Specific messages can be sent to the BBI Journal from any TSO/E REXX or CLIST application where specific information about an automation situation or multi-system support can be provided via XCF connectivity.

AOAnywhere opens up automation possibilities through a simple command processor that can be invoked in a variety of environments.

Installation Requirements

To use the AOEXEC command processor under TSO/E, it must be available to the TSO/E user under the STEPLIB/LINKLIB concatenation. Otherwise the command processor will not be found.

Currently, you can secure access to AOAnywhere with the same security measures available for writing AutoOPERATOR EXECs. These security measures are described in the BMC Software document *Implementing Security for MAINVIEW Products*.

API Implementation under REXX and CLIST

The API functions are available as a separate command processor. This feature allows simultaneous use for the API by TSO/REXX and TSO/CLIST.

Differences between IMFEXEC and AOEXEC Parameter Syntax

Parameters and return codes between IMFEXEC and AOEXEC commands are identical with the following exceptions:

- All AOEXEC variable operations (VPUT, VGET, VDEL and their long counterparts) specify the variable names using the VAR() keyword instead of a positional parameter.
- The AOEXEC VPUT command has no FROM(), USING() or ENCRYPT() parameters.
- The AOEXEC VGET command has no INTO(), DECRYPT or DELIM() parameters.
- All AOEXEC commands might return a return code of –1 with the TGTSS() keyword. In this case, either the request timed out or the target system was shut down in the middle of a request.
- All AOEXEC commands accommodate two extra keywords, SS | SSID() and TGTSS() where

SS | SSID(subsystem identifier)	Required keyword. SS | SSID( ) specifies the subsystem identifier of a local subsystem. If the TGTSS() keyword is not specified, this SSID is the subsystem where the requested function is executed.
TGTSS(target system identifier)	Optional keyword. If the TGTSS() keyword is specified, the subsystem specified by the SS | SSID() keyword is considered a router and the actual function is executed on the subsystem specified by TGTSS(). It must be in the same sysplex as the BBI-SS specified with the SSID() keyword, and both systems must have the same XCFGROUP specified in the BBPARM BBISSPxx.

Additional Differences

All AOEXEC commands return values in return codes that are listed with each AOEXEC command. The values are returned differently depending on where the AOEXEC command is issued from. For example, if the AOEXEC command is used in a REXX EXEC, the return code will be returned in the RC variable. If the AOEXEC command is used in a CLIST EXEC, the return code is returned in &LASTCC.

Furthermore, the AOEXEC command processor attempts to streamline the syntax of some of the supported commands. For example:

- For the AOEXEC ALERT command, the first two positional parameters are replaced by the keywords TEXT() and KEY() respectively.

- The TARGET() keyword has been removed from all AOEXEC commands and replaced by the TGTSS() keyword.
- The VAR() keyword can be overwritten in the invoked EXEC by specifying the IMFEXEC SHARE command.
- The AOEXEC SELECT command has a new keyword, VAR(). This keyword specifies the names of any number of variables that will be exchanged with the LOCAL variable pool of the selected EXEC.

  Before the target EXEC begins, the contents of these variables are placed as variables of the same name into the LOCAL pool.

  When the EXEC ends, the contents of these variables in the target EXEC's LOCAL pool are extracted again and placed as TSO variables in the pool of the invoking EXEC.

Each of these exceptions has been reflected in the documentation for each of the AOEXEC commands.

Example

Here is an example about how to share variables between an EXEC running in a TSO/E address space and an EXEC running in the subsystem:

```
a='ONE'
"AOEXEC SELECT EXEC(DEMO) VAR(A B) WAIT(YES) SSID(TGTA)"
say b
```

These lines within a REXX EXEC causes the EXEC named DEMO to be invoked on the subsystem named TGTA. Before the EXEC begins processing the contents of the variables of the invoking EXEC, variables A and B are placed in the EXEC's LOCAL variable pool. Variable B's value has not been set but specified for data exchange with the PAS EXEC.

The code for the EXEC in the subsystem is

```
"IMFEXEC VGET A LOCAL"
"IMFEXEC MSG A"
b='TWO'
"IMFEXEC VPUT B LOCAL"
```

This code causes the message ONE to be written to the subsystem journal. Subsequently the value of TWO is placed into the variable B and this variable placed into the EXEC's LOCAL variable pool. This variable pool will be transmitted back to the invoking REXX EXEC. Note that the specified contents of the LOCAL variable pool, in this case the variables A and B, are shared with the invoking EXEC.

The statement

```
say b
```

causes the value of B, in this case now TWO, to be written to the TSO/E console.

Implementing the AOAnywhere Batch Interface: AOSUBX

For batch jobs, AOAnywhere contains a facility called AOSUBX. This facility is a partial replacement for the existing IMFSUBEX function. Both functions allow you to invoke an EXEC from a batch step with the PARM= specification.

Both facilities allow for requests to be scheduled across systems. However, AOSUBX (unlike IMFSUBEX), requires sysplex connectivity between the systems.

In addition, AOSUBX offers significantly shorter execution time and the ability to wait for EXECs scheduled across systems without tying up the VTAM link between multiple BBI-SS PASs. When sysplex connectivity exists, (as it should when using AOSUBX), you should always choose AOSUBX for EXEC invocation.

In a TSO/E environment, a suite of command processor functions is available under the AOEXEC facility. Refer to "AOEXEC Commands" on page 135 for details. This approach is preferred under TSO/E as opposed to using the AOSUBX facility.

Why Use AOSUBX

Under certain conditions it is convenient to initiate an EXEC from a jobstep or procstep. This invocation can signal the completion of a particular function or the necessity to execute AutoOPERATOR functions on behalf of the step. At times these functions need to be executed before the job or process can continue and some sort of completion indication needs to be passed back and forth between the invoked EXEC and the step.

AOSUBX (like IMFSUBEX) meets these requirements. It represents a high speed path to EXEC invocation on either local or remote systems and allows the caller to wait for the completion of this EXEC, returning the exit code of the invoked EXEC as a modified return code.

Syntax

The general syntax for invoking AOSUBX from a jobstep is as follows:

```
//STEPX EXEC PGM=AOSUBX,PARM='parms…'
//STEPLIB DD DISP=SHR,DSN=prefix..BBLINK
```

The `parms` entered must specify the EXEC() and SS | SSID() keywords whereas the TGTSS() and WAIT() keywords are optional. The description of the keywords follows.

Keyword	Required/ Optional	Description
EXEC	Required	Specifies the name of the EXEC and any parameters to be passed to the symbolic variables defined as input in the EXEC. Maximum length is any number of characters allowed by the PARM= statement. Note that the SS | SSID() parameter is required.
SS | SSID()	Required	Specifies a BBI-SS PAS to process this EXEC or the name of a local BBI-SS PAS that will route the request to a remote BBI-SS PAS (as specified by the TARGET|TGTSS() keyword).

Keyword	Required/ Optional	Description
TARGET|TGTSS()	Optional	Specifies the name of a remote BBI-SS PAS where the request is to be routed. Sysplex connectivity between the local and remote BBI-SS PAS must be available. The target BBI-SS PAS must be in the same sysplex as the BBI-SS specified with the SSID() keyword, and both systems must have the same XCFGROUP specified in the BBPARM BBISSPxx.
WAIT()	Optional	Specifies whether to wait for the completion of the EXEC or continue after scheduling. Note that WAIT(Y) is required to obtain the exit code of the EXEC.

Return codes are listed in the following table.

Value	Description
-1	When the TGTSS() keyword is used, indicates that either the request timed out or the target system was shut down in the middle of a request.
0	Command was executed successfully.
8	EXEC you are trying to invoke does not exist.
16	Syntax error occurred.
32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as originating subsystem.
36	The local BBI-SS PAS specified by the SSID parameter is not available.
40	Security definitions disallowed access to this function on the specified subsystem.
48	Incompatible AOAnywhere versions exist.
52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.
2048 + return code	When specifying WAIT(Y) and the command was executed successfully, a value 2048 is added to the EXEC's exit code before the return code is generated.

Examples

```
EXEC(TEST A B C D()) SSID(RE61) TGTSS(RE62) WAIT(Y)
```

An EXEC with the name of `TEST` will be invoked, passing the parameters `A B C D()`. Note that parentheses are allowed. A BBI-SS PAS with the SSID of `RE61` must be active on the same MVS image that will route the request to another BBI-SS PAS with an SSID of `RE62`.

The step will wait until the EXEC ends and a return code of 2048 plus the exit code of the EXEC is returned. For example, if the EXEC ended with an IMFEXEC EXIT CODE(12), the step receives a return code of 2060 (2048+12).

AOEXEC Commands

The following table lists the IMFEXEC AOEXEC commands and the page where you can find more information.

Command	Page	Function
AOEXEC ALERT	137	Creates and manages exception messages and message queues that can be displayed by any of the STATUS applications and ALERT Management Facility applications.
AOEXEC MSG	161	Logs a message in the BBI-SS PAS Journal log.
AOEXEC NOTIFY	163	Sends a request through AutoOPERATOR to issue a pager call using the MAINVIEW AutoOPERATOR Elan Workstation component (if it is installed).
AOEXEC SELECT	165	Invokes an EXEC or a program.
AOEXEC SYSINFO	167	Searches the current MVS image for an AutoOPERATOR subsystem that runs AOAnywhere support.
AOEXEC VDEL	171	Deletes one or more variables from one of the AutoOPERATOR variable pools.
AOEXEC VGET	174	Copies one or more variables from one of the AutoOPERATOR pools into the EXECs function pool.
AOEXEC VLST	176	Lists variable names defined in the AutoOPERATOR pools.
AOEXEC VPUT	179	Copies one or more variables from the EXECs function pool into one of the AutoOPERATOR pools.
AOEXEC VDELL	181	Deletes one or more long variables from one of the AutoOPERATOR variable pools.
AOEXEC VGETL	183	Copies one or more long variables from one of the AutoOPERATOR pools into the TSO pool.
AOEXEC VLSTL	185	Retrieves a long variable from the specified pool and places it into the TSO pool.
AOEXEC VPUTL	187	Creates or sets a long variable from a variable in the TSO pool.

General Coding Conventions

The following sections briefly describe the coding conventions for using the AOEXEC command statements.

The command syntax is the keyword AOEXEC, followed by the command and any necessary parameters; for example:

```
AOEXEC command [parameters]
```

Using Variable Names

Variable names are limited to 32 characters in length except where noted. The first character of the variable must be alphanumeric or one of the following special characters:

- $
- @
- #

Reading Return Codes

All AOEXEC commands return values in return codes that are listed with each AOEXEC command. The values are returned differently depending on where the AOEXEC command is issued from. For example, if the AOEXEC command is used in a REXX EXEC, the return code will be returned in the RC variable. If the AOEXEC command is used in a CLIST EXEC, the return code is returned in &LASTCC.

Understanding Command Statement Syntax

Each AOEXEC command statement description includes a table describing the parameters for the command. The table uses the following format:

Parameter	Function	Notes
1	**2**	**3**

The numbers in this table correspond to the following descriptions:

1 A short parameter identifier. If the parameter has uppercase letters, this identifier must be coded exactly as shown.

If parts of the identifier are shown in **bold**, this parameter can be abbreviated, using the bold letters.

Positional parameters are not associated with a specific identifier. In these cases, this column contains an alias that describes the parameter.

2 The function of the parameter.

3 Notes about the parameter. Typically, these notes describe any length, value, range, or string limitations.

AOEXEC ALERT

This command manages exception messages and message queues that can be displayed by any of the STATUS applications and ALERT Management Facility applications.

Command	Parameters
AOEXEC ALERT	[KEY()] [TEXT('text string')] [ALARM(YES|NO)] [COLOR(RED|PINK|YELLOW|DKBLUE|LTBLUE|GREEN|WHITE)] [DISPOSE(KEEP|DELETE)] [ESCALATE(UP|DOWN)] [ESCEXEC('execname p1 p2 p3 ... pn')] [EXEC('execname p1 p2 p3 ... pn')] [FUNCTION(ADD|COUNT|CREATEQ|DELETE|DELETEQ|LISTQ|READQ)] [HELP(panelname)] [INTERVAL(nnnn,nnnn,nnnn,nnnn,nnnn,nnnn)] [PCMD('cmd string')] [POSITION(position)] [PRI(CRITICAL|MAJOR|MINOR|WARNING|INFORMATIONAL|CLEARING)] [PUBLISH(REPLACE|ADD|NO] [QUEUE(MAIN|queue name)] [RETAIN(YES|NO)] SS | SSID(subsystem identifier) [SYSTEM(YES|NO)] [TGTSS(target subsystem identifier)] [ORIGIN(origin)] [UDATA('user data')] [USER(user name)]

The following table describes the parameters.

Parameter	Function	Notes
KEY	The key used to uniquely identify an ALERT within a queue	Maximum length is 64 alphanumeric positions. Required for FUNCTION(ADD) FUNCTION(DELETE) Optional for FUNCTION(READQ) You must specify a unique key for every ALERT you create. If you create a second ALERT with the same key as an already existing ALERT in the queue, the second ALERT will overwrite the first ALERT.
TEXT	The text of the ALERT message	Maximum message length is 255 alphanumeric positions. Required for: FUNCTION(ADD) If the contents of the text are null but specified (for example, zero length), the ALERT text is replaced by N/A. A specification of /N within the alert text forces a line break. You must include a blank space before and after using /N. This parameter applies also to the READQ and COUNT functions. Only ALERTs matching this text string are considered during these operations.
ALARM	An audible alarm emitted from the terminal on the ALERT Detail application	Possible values are YES Sound alarm. NO Do not sound alarm. NO is the default.

Parameter	Function	Notes
COLOR|COL	The color in which the ALERT is displayed in the ALERT DETAIL and STATUS applications (overrides default color associated with ALERT priority)	This parameter does not have any impact upon the ALERT OVERVIEW application. When an ALERT's priority is increased or decreased (with the ESCALATE parameter), the new ALERT priority's color will change to the color in the following list: RED - CRITICAL PINK - MAJOR YELLOW - MINOR DKBLUE - WARNING LYBLUE - INFORMATIONAL GREEN - CLEARING
DISPOSE	Allows you to specify whether an ALERT is kept or deleted when it has reached its final escalation priority level	This keyword must be used with the INTERVAL keyword. Possible values are KEEP — Keep the ALERT in its queue. DELETE — Delete the ALERT from the queue. KEEP is the default. The variable AMFEDISP returns the value of this keyword.
ESCALATE	Allows you to create ALERTs that can change in priority over a specified interval of time	This keyword must be used with the INTERVAL keyword. Possible values are UP — The ALERT priority is upgraded from less critical to more critical. DOWN — The ALERT priority is downgraded from more critical to less critical. UP is the default. The variable AMFEDIR returns the value of this keyword.

Parameter	Function	Notes
ESCEXEC	Allows you to specify an EXEC (with parameters) that is scheduled when the ALERT reaches its final priority level	This keyword must be used with the INTERVAL keyword. The variable AMFEEXEC returns the value of this keyword.
EXEC	The name of the ALERT-initiated follow-up EXEC and its parameters	Maximum length is 256 characters. Refer to “Parameters Passed to the EXEC” on page 29 for more information about parameters passed to ALERT-initiated EXECs.
FUNCTION|FUN	The function to be performed	Use the FUNCTION keyword with • ADD • COUNT • CREATEQ • DELETE • DELETEQ • LISTQ • READQ For more information about these functions and the return codes they generate, refer to Table 8 on page 146.
HELP	The name of an extended help panel	Maximum length is 8 characters. This help panel is displayed when you enter the EXPAND primary command in the ALERT DETAIL application while the cursor is positioned on the ALERT. The help panel is a text member without any formatting or control characters. The help text member must be included the BBPLIB concatenation for the terminal session.

Parameter	Function	Notes
INTERVAL	Allows you to specify one to six intervals of time over which the priority of an ALERT will change An ALERT's priority can either increase (become more critical) or decrease (become less critical) in priority over the specified time intervals. The interval can be specified from 0 to 9999 minutes. At least one interval must be specified for an ALERT when ESCALATE is specified. When the final interval expires • The action specified by the DISPOSE keyword occurs (either the ALERT is deleted or kept) • If an EXEC is specified with the ESCEXEC keyword, the EXEC is scheduled	This keyword **must be used** with the ESCALATE keyword and you must specify at least one interval for an ALERT when ESCALATE is specified. The variables AMFEINT1 through AMFEINT6 return the values associated with this keyword. In addition, when you want to have an ALERT change in priority, you must always code one interval more than the number of changes. No priority changes occur in the last interval. For example, if you want an ALERT to change from MAJOR to CRITICAL, you must code two interval periods. Refer to "Examples of ALERT Escalation" on page 156 for examples.
ORIGIN	A new origin to assign to this ALERT	A 1- to 8-character user-defined origin that is assigned to the ALERT. The first character cannot be a numeric. This user-defined origin overrides the EXEC's IMFSYSID (or the originating job name for the EXEC).

Parameter	Function	Notes
PCMD	A command to be executed if the terminal operator uses the TRANSFER command on the ALERT DETAIL panel	Any command that is valid from the COMMAND line is a valid value for this parameter. Maximum length is 256 characters. PCMD is executed as if it were entered on the COMMAND line. You should use the SYSTEM parameter (described below) or include the BBI SYSTEM command for ALERTs that contain PCMD to ensure that the target field of the transferred-to application will be correct. If you use the SYSTEM parameter, the SYSTEM command is executed after all other commands specified with PCMD have executed. For example: `PCMD('CICS;EX TRAN;SYSTEM SYSA')` Note that if you have blanks in the PCMD statement, you must use single quotation marks.
POSITION|POS	The order of the ALERT in the queue to read	Valid values are in the range from 1 to 32,767. This parameter is used only with the READQ function.
PRIORITY	The priority of the ALERT	A valid value is one of the following options: CRITICAL MAJOR MINOR WARNING INFORMATIONAL CLEARING

Parameter	Function	Notes
PUBLISH	Specifies whether an ALERT is published and how it is published to connected PATROL Enterprise Manager (PATROL EM) workstations that have subscribed to receive ALERTs through the General Message Exchange (GME).	Possible values are REPLACE — An ALERT REPLACE command for the ALERT's key/queue is sent to all PATROL EM workstations that have subscribed to receive ALERTs from this AutoOPERATOR. If there is already an ALERT with that key/queue on a PATROL EM workstation, it is deleted before writing the new ALERT with that key/queue. ADD — An ALERT ADD command is sent to all workstations that have subscribed to receive ALERTs from this AutoOPERATOR. If there is already an ALERT with that key/queue on a PATROL EM workstation, it is not deleted before writing the new ALERT with that key/queue. ADD is the default. NO — The ALERT is not written to the connected PATROL EM workstations even if they have subscribed to receive ALERTs.
QUEUE|QUE	The name of the queue to access or into which to place the ALERT	Length can be 1 - 8 characters; embedded blanks are valid.

Parameter	Function	Notes
RETAIN	Allows you to specify that an ALERT will be retained across BBI-SS PAS restarts (both cold and warm restarts) and MVS IPLs. Note that using this parameter causes the ALERT to be written to DASD. Therefore, you should use this parameter only after careful consideration. A BBI-SS PAS (warm or cold) start or MVS IPL might eliminate the exceptional situation that caused the ALERT in the first place.	Possible values are YES Retain this ALERT in disk space so that it can survive a BBI-SS PAS warm or cold start. NO Do not retain this ALERT to survive BBI-SS PAS warm or cold starts. NO is the default. ALERTs that specify RETAIN(YES) cannot also specify the INTERVAL keyword. In other words, ALERTs that are to be retained across BBI-SS PAS restarts or MVS IPLs cannot change priority (either increase or decrease). The variable AMFRTAIN returns the value of this keyword.
SS | SSID	SS | SSID() specifies the subsystem identifier of a local subsystem. If TGTSS() is not specified, this is the subsystem where the requested function is executed.	Required keyword.
SYSTEM	Determines whether the ALERT DETAIL processor switches the current target to the origin of the ALERT when processing a TRANSFER (PCMD).	The default is YES, switch the current target to the origin of the ALERT when processing a TRANSFER (PCMD). NO specifies do not switch current target to the origin of the ALERT when processing a TRANSFER (PCMD). The target is changed to reflect what was coded in the ORIGIN parameter or the AutoOPERATOR SSID.
TGTSS	If the TGTSS() keyword is specified, the subsystem specified by the SS | SSID() keyword is considered a router and the actual function is executed on the subsystem specified by TGTSS(). If TGTSS() is not specified, the requested function is executed on the subsystem specified by the SS | SSID keyword.	Optional keyword. It must be in the same sysplex as the BBI-SS specified with the SSID() keyword, and both systems must have the same XCFGROUP specified in the BBPARM BBISSPxx.

Parameter	Function	Notes
UDATA	Any desired user data string	Maximum length is 256 bytes The contents of the UDATA field can be retrieved using the READQ function.
USER	The name of the user ID to which the ALERT is addressed	A 1 - 8 character valid BBI-TS user ID. Contents of the user field can be used to tailor ALERT DETAIL displays using the ALERT DETAIL PROFILE panel. Refer to the “ALERT Management Facility” chapter in the *MAINVIEW AutoOPERATOR Basic Automation Guide* for more information.

Return Codes for FUNCTION Keywords

The following table lists and describes in alphabetical order the return codes for the different functions that can be used with the FUNCTION keyword in an AOEXEC ALERT EXEC statement.

Table 8. FUNCTION Names and Return Codes

FUNCTION	Description	Return Code Value	Return Code Description
ADD	Adds an ALERT to a queue and creates a new queue if one does not already exist	-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
		0	ADD was successful.
		16	Invalid syntax used.
		20	ALERT queue is full.
		32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as the originating subsystem.
		36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
		40	Security definitions disallowed access to this function on the specified subsystem.
		48	Incompatible AOAnywhere versions exist.
		52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Table 8. FUNCTION Names and Return Codes (Continued)

FUNCTION	Description	Return Code Value	Return Code Description
COUNT	Counts the numbers of ALERTs in a given queue. Refer to “TSO Variables Returned from COUNT” on page 154 for more information.	-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
		0	COUNT was successful; count value is returned in variable AMFCOUNT.
		8	Queue does not exist.
		16	Invalid syntax used.
		32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as the originating subsystem.
		36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
		40	Security definitions disallowed access to this function on the specified subsystem.
		48	Incompatible AOAnywhere versions exist.
		52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Table 8. FUNCTION Names and Return Codes (Continued)

FUNCTION	Description	Return Code Value	Return Code Description
CREATEQ	Creates a new ALERT queue.	-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
		0	Queue was created successfully.
		4	Queue already exists.
		16	Invalid syntax used.
		32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as the originating subsystem.
		36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
		40	Security definitions disallowed access to this function on the specified subsystem.
		48	Incompatible AOAnywhere versions exist.
		52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Table 8. FUNCTION Names and Return Codes (Continued)

FUNCTION	Description	Return Code Value	Return Code Description
DELETE	Deletes an ALERT by the ALERT key.	-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
		0	DELETE was successful.
		4	ALERT does not exist.
		8	Queue does not exist.
		16	Invalid syntax used.
		32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as the originating subsystem.
		36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
		40	Security definitions disallowed access to this function on the specified subsystem.
		48	Incompatible AOAnywhere versions exist.
		52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Table 8. FUNCTION Names and Return Codes (Continued)

FUNCTION	Description	Return Code Value	Return Code Description
DELETEQ	Deletes an ALERT queue.	-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
		0	DELETEQ was successful.
		4	Queue does not exist.
		16	Invalid syntax used.
		32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as the originating subsystem.
		36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
		40	Security definitions disallowed access to this function on the specified subsystem.
		48	Incompatible AOAnywhere versions exist.
		52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Table 8. FUNCTION Names and Return Codes (Continued)

FUNCTION	Description	Return Code Value	Return Code Description
LISTQ	Lists (in TSO variable IMFNOL) the number of ALERT queues present in the target subsystem. Refer to "TSO Variables Returned from LISTQ" on page 154 for more information.	-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
		0	LISTQ was successful; ALERT queue data is returned.
		16	Invalid syntax used.
		32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as the originating subsystem.
		36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
		40	Security definitions disallowed access to this function on the specified subsystem.
		48	Incompatible AOAnywhere versions exist.
		52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Table 8. FUNCTION Names and Return Codes (Continued)

FUNCTION	Description	Return Code Value	Return Code Description
READQ	Reads an ALERT from the queue and returns the characteristics of the ALERT in TSO variables. Refer to “TSO Variables Returned from the READQ Parameter” for more information.	-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
		0	READQ was successful; ALERT data returned.
		4	Either no match was found when using KEY and TEXT criteria or the search ran past the end of the queue when using the POSITION keyword.
		8	Queue does not exist.
		16	Invalid syntax used.
		32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as the originating subsystem.
		36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
		40	Security definitions disallowed access to this function on the specified subsystem.
		48	Incompatible AOAnywhere versions exist.
		52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

TSO Variables Returned from the READQ Parameter

The following table lists the TSO variables returned from the READQ parameter.

Name	Contents	Maximum Length/Format	Example
AMFALARM	Alarm value of the alert	1 / Y (YES) or N (NO)	Y
AMFCOLOR	Color of ALERT	6 / As specified by COLOR parameter	RED

Name	Contents	Maximum Length/Format	Example
AMFEDIR	Increase or decrease the priority of the ALERT when it is escalated	1 / Character U (up) or D (down)	D
AMFEDISP	Keep or delete the ALERT at the final escalation level	1 / Character (K or D)	K
AMFEEXEC	Name of EXEC and EXEC parameters scheduled at final escalation priority	0-256 / Character	ALRTEXEC
AMFEINT1 AMFEINT2 AMFEINT3 AMFEINT4 AMFEINT5 AMFEINT6	Number (in minutes) from 0 to 9999	4 / Numeric (or null)	15
AMFEXEC	EXEC and EXEC parameters associated with the ALERT	0-256 / Character	DBSTART SHIFT2
AMFHELP	Extended Alert member name	8 / Character	HELPXT2
AMFIDATE	Date ALERT was issued	9 / DD-MMM-YY	14-FEB-92
AMFITIME	Time ALERT was issued	8 / hh:mm:ss	12:02:24
AMFKEY	Key of the ALERT	1-64 / Character	DASD01
AMFORGN	Origin of ALERT	1-8 / Character	CICSPROD
AMFPCMD	Primary command specified in ALERT	0-256 / Character	CICS; EX TRAN
AMFPRIOR	Priority of ALERT	13 / As specified in PRIORITY parameter	INFORMATIONAL
AMFPSYS	Value for SYSTEM keyword (could be either YES or NO)	1 / Character (Y or null)	Y
AMFPUB	Value of the PUBLISH keyword when an ALERT is created	2-7/ADD, REPLACE, or NO	ADD
AMFQUEUE	Name of queue for ALERT	8 / Character	MAIN
AMFRTAIN	Specifies whether to retain an ALERT across BBI-SS PAS warm and cold starts	1 / Character (Y or N)	Y
AMFSSID	System from which ALERT was issued	8 / Character	SYSB
AMFTEXT	Text of the ALERT	0-255 / Character	This ALERT is a test

Name	Contents	Maximum Length/Format	Example
AMFTGT	Target to which ALERT was issued	1-8 / Character	IMS22P
AMFUDATA	User data string	0-256 / Character	Any value specified in UDATA parameter
AMFUSER	Name of the user ID to which the ALERT is addressed	8 / Character	JDB1

TSO Variables Returned from COUNT

The following table lists the TSO variables returned from the COUNT parameter.

Name	Contents
AMFCOUNT	Number of ALERTs in designated queue

TSO Variables Returned from LISTQ

The following table lists the TSO variables returned from the LISTQ parameter.

Name	Contents
IMFNOL	Number of queues present in the target subsystem. In variables LINE1 through LINExxx, it returns the names of the all the queues. Limit is 500 queue names.

Examples

This section describes examples using the AOEXEC ALERT command. A brief discussion follows each example.

Example 1: Creating a Multiline ALERT.

```
"AOEXEC ALERT KEY(NETW2) ",
  "TEXT('COMMUNICATION LINES DOWN: /N - DALLAS /N - CHICAGO') ",
  "FUNCTION(ADD) QUEUE(NETWORK) ",
  "PRIORITY(CRITICAL) COLOR(PINK) SSID(RE61)"
```

ALERTs are created as single-line messages unless you use the characters `/N` in the alert text parameter. The characters `/N` indicate the beginning of a new line of alert text.

You must use a blank space before and after `/N`. In the example above, the alert text parameters includes the use of `/N` in two places. The EXEC command in this example produces the following multiline ALERT:

```
___ 11:43     CHICAGO  COMMUNICATION LINES DOWN:
                                - DALLAS
                                - CHICAGO
```

Example 2: Associating a Help Panel with an ALERT.

```
"AOEXEC ALERT KEY(NETW1) ",
  "TEXT('ALM0100 - 8100 COMMUNICATION LINE DOWN: /N - CHI998A21') ",
  "FUNCTION(ADD) QUEUE(NETWORK) PRIORITY(WARNING) HELP(H8100) ",
   "COLOR(RED) SSID(RE61)"
```

Use the HELP keyword of the AOEXEC ALERT command statement to indicate there is a help panel associated with an ALERT.

Prior to using the HELP keyword in the AOEXEC ALERT command, you must create and add the help panel to BBPLIB. The HELP keyword specifies the name of the BBPLIB member name. The example shows an AOEXEC ALERT command statement that specifies a help panel named H8100. The example is a REXX statement and therefore uses double quotation marks. The ALERT created by the EXEC appears on the ALERT DETAIL panel in the following format:

```
TIME  IND ORIGIN _____________________________________
11:44 h   CHICAGO  ALM0100 8100 COMMUNICATION LINE DOWN:
                          -CHI998A21
```

The ALERT is displayed with an `h` in the `IND` column. This `h` indicates that there is a help panel associated with the ALERT.

To access the help panel, place the cursor anywhere on the ALERT text and press the PF key assigned to EXPAND. You can also type EXPAND on the COMMAND line and then place the cursor anywhere on the ALERT text and press ENTER.

Example 3: Managing ALERT Queues.

```
/* REXX */
"AOEXEC VGET VAR(THRSHOLD) SSID(RE61)"
"AOEXEC ALERT FUNCTION(COUNT) QUEUE(NETWORK) SSID(RE61)"
n=amfcount
do while n > 0
  "AOEXEC ALERT FUNCTION(READQ) QUEUE(NETWORK) POSITION("N") SSID(RE61)"
  if rc=0 then do
    if amfudata > thrshold then do
      "AOEXEC ALERT KEY("amfkey") FUNCTION(DELETE) QUEUE(NETWORK) SSID(RE61)"
      "AOEXEC ALERT KEY("amfkey") FUNCTION(ADD) TEXT('"amftext"') QUEUE(SUPERVSE)",
      "SSID(RE61)"
    END
  END
  n = n - 1
END
```

You can periodically check the queues for ALERTs that have not been responded to and escalate their priority.

In the above EXEC, the READQ function is used to set AMFCOUNT equal to the number of ALERTs in the NETWORK queue. The EXEC then reads each ALERT from the NETWORK queue using POSITION and tests the user data presented in the AMFUDATA variable.

If the criteria is met, the ALERT is deleted from the NETWORK queue using the AMFKEY variable (the key of the ALERT). Then the ALERT is added to the supervisor's queue using the same key and using the original text in the AMFTEXT variable.

Note: This example assumes that the ALERTs were originally created with some meaningful user data (such as the date and time).

Examples of ALERT Escalation

The following examples show how to create ALERTs with the ESCALATE parameter so that an ALERT can increase or decrease in priority over specified intervals of time.

Example 1: Escalating an ALERT from lowest to highest priority: The ALERT in this example will be upgraded from Informational to Critical priority over five intervals. The following list describes the properties of the ALERT:

- The original priority of the ALERT is Informational (`PRIORITY(info)`).
- The ALERT's priority will be upgraded (`Escalate(up)`).
- The priority will be upgraded gradually over the intervals of 10 minutes, 20 minutes, 30 minutes, 30 minutes, and 40 minutes (`Interval(10,20,30,30,40)`).
- When the ALERT reaches the final priority level, the ALERT should be deleted (`Dispose(delete)`).

```
"AOEXEC ALERT KET(KEY1) TEXT('test alert') PRIORITY(INFO) ESCALATE(UP)" ,
  "INTERVAL(10,20,30,30,40) DISPOSE(DELETE) SSID(RE61)"
            1  2  3  4  5
```

When the EXEC that is associated with this ALERT is scheduled, the ALERT's original priority is Informational. After 10 minutes (**1**), the priority is upgraded automatically from Informational to Warning. The ALERT stays at the

Warning priority for 20 minutes (**2**) and is upgraded to Minor. The ALERT stays at Minor priority for 30 minutes (**3**) before being upgraded to Major. It stays at Major priority for 30 minutes (**4**) before being upgraded to Critical. After remaining at Critical for 40 minutes (**5**), the ALERT is deleted.

Example 2: Downgrading ALERT priority over two intervals: The ALERT in this example will be downgraded over two intervals. The following list describes the properties of the ALERT:

- The original priority of the ALERT is Minor (`PRIORITY(MINOR)`).
- The ALERT's priority will be downgraded (`ESCALATE(DOWN)`).
- The priority will be downgraded over the intervals of 10 minutes and 20 minutes (`INTERVAL(10,20)`).
- When the ALERT reaches the final priority level, the ALERT should be deleted (`DISPOSE(DELETE)`).

```
"AOEXEC ALERT KEY(KEY2) TEXT('test alert') PRIORITY(MINOR) ESCALATE(DOWN) " ,
  "INTERVAL(10,20) DISPOSE(DELETE) SSID(RE61)"
            1  2
```

When the EXEC that is associated with this ALERT is scheduled, the ALERT's original priority is Minor. After 10 minutes (**1**), the priority is downgraded automatically from Minor to Warning. The ALERT remains at the Warning priority for 20 minutes (**2**) and is deleted at the end of the interval.

The intervals in this example also can be validly coded as follows:

```
INTERVAL(10,20,)
```

or

```
Interval(10,20,,)
```

or

```
Interval(10,20,,,,)
```

Example 3: Upgrading an ALERT and scheduling an escalation EXEC: The ALERT in this example will be upgraded over two time intervals and, at the end of the second interval, an escalation EXEC will be scheduled. The following list describes the properties of the ALERT:

- The original priority of the ALERT is Minor (`PRIORITY(MINOR)`).
- The ALERT's priority will be upgraded (`ESCALATE(UP)`).
- The priority will be upgraded over the intervals of 10 minutes and 20 minutes (`INTERVAL(10,20)`).
- When the ALERT reaches the final priority level, the ALERT should be kept until it is manually deleted (`DISPOSE(KEEP)`).
- When the ALERT completes its final interval, an EXEC named E100 with three parameters is scheduled (`ESCEXEC('E100 p1 p2 p3')`).

```
"AOEXEC ALERT KEY(KEY2) TEXT('test alert') PRIORITY(MINOR) ESCALATE( UP) ",
"INTERVAL(10, 20) DISPOSE(KEEP) ESCEXEC('E100 p1 p2 p3')"
            1   2
```

When the EXEC that schedules this ALERT is scheduled, the ALERT's original priority is Minor. After 10 minutes (**1**), the priority is upgraded automatically from Minor to Major. The ALERT remains at the Major priority for 20 minutes (**2**) and the EXEC e100 with its three parameters is scheduled at the end of the interval. The ALERT remains at the Major priority until it is manually deleted.

Example 4: Skipping ALERT priorities during ALERT escalation: The ALERT in this example will be upgraded from Informational to Major while skipping the intermediate ALERT priorities. The following list describes the properties of the ALERT:

The original priority of the ALERT is Informational (`PRIORITY(INFO)`).

- The ALERT's priority will be upgraded (`ESCALATE(UP)`).
- The priority will be upgraded over the two intervals of 10 and 20 minutes.

  However, to skip ALERT priorities, you must specify an interval of zero minutes for each of the intervals you want to skip.

In this example, the ALERT will skip two priorities and change from Informational priority directly to Major after a 10-minute interval (`INTERVAL(10,0,0,20)`).

- When the ALERT reaches the final priority level, the ALERT should be kept until it is manually deleted (`DISPOSE(KEEP)`).
- When the ALERT completes its final interval of 20 minutes, an EXEC named E100 with three parameters is scheduled (`ESCEXEC('E100 p1 p2 p3')`).

```
"AOEXEC ALERT KEY(KEY2) TEXT('test alert') PRIORITY(INFO) ESCALATE(UP) ",
"INTERVAL(10,0,0,20) DISPOSE(KEEP) ESCEXEC('E100 p1 p2 p3') SSID(RE61)"
          1 2 3 4
```

When the EXEC that schedules this ALERT is scheduled, the ALERT's original priority is Informational. After 10 minutes (**1**), the ALERT's priority is upgraded automatically from Informational to Major. To skip the intermediate priorities, you must code zero minutes for both Warning and Minor priorities (**2** and **3**).

The ALERT remains at the Major priority for 20 minutes (**4**) and the EXEC e100 with its three parameters is scheduled at the end of the interval. The ALERT remains at the Major priority until it is manually deleted.

The intervals in this example also can be validly coded as follows:

```
INTERVAL(10,0,0,20,)
```

or

```
INTERVAL(10,0,0,20,,)
```

Example 5: Showing the elapsed time for an escalated ALERT. The ALERT in this example will be upgraded from Minor to Major in one 10-minute interval. The following list describes the properties of the ALERT:

- The original priority of the ALERT is Minor (`PRIORITY(MINOR)`).
- The ALERT's priority will be upgraded (`ESCALATE(UP)`).
- The priority will be upgraded over one interval of 10 minutes (`INTERVAL(10)`).
- When the ALERT reaches the final priority level, the ALERT should be deleted (`DISPOSE(DELETE)`).

- When the ALERT completes its final interval, an EXEC named E100 with three parameters is scheduled (ESCEXEC('E100 p1 p2 p3')).

```
"AOEXEC ALERT KEY(KEY2) TEXT('test alert') PRIORITY(MINOR) ESCALATE(UP) ",
  "INTERVAL(10, 20) DISPOSE(DELETE) ESCEXEC('E100 p1 p2 p3') SSID(RE61)"
```

The following example shows the life of the ALERT over time:

```
1:00pm                    1:10pm                          1:30pm
A Minor ALERT       -->   The ALERT is upgraded   -->     The ALERT is deleted
is created                to Major Priority               and the EXEC e100
                                                          is scheduled
The ALERT stays at this   The ALERT stays at this
priority for 10 minutes    priority for 20 minutes
```

Examples of Invalid Coding with the Interval Parameter

Some examples of invalid coding are as follows:

Example 1: The interval keyword must contain at least one value.

```
"AOEXEC ALERT KEY(KEY4) TEXT('test alert') PRIORITY(MAJOR) ESCALATE(UP) ",
  "INTERVAL(, 10, 10) SSID(RE61)"
```

Example 2: You can only specify as many intervals as there are between an originating priority and the end priority.

```
"AOEXEC ALERT KEY(KEY4) TEXT('test alert') PRIORITY(INFO) ESCALATE(UP)",
  "INTERVAL(, 10,,20) SSID(RE61)"
```

In example 2, there is only one priority that a major ALERT can be upgraded to (Critical) and yet three intervals are specified.

Example 3: The interval keyword cannot have null values for intervals.

```
"AOEXEC ALERT KEY(KEY4) TEXT('test alert') PRIORITY(MAJOR) ESCALATE(UP)"
  "INTERVAL(,10,10) SSID(RE61)"
```

or

```
"AOEXEC ALERT KEY(KEY4) TEXT('test alert') PRIORITY(INFO) ESCALATE(UP)"
 "INTERVAL(,10,,20) SSID(RE61)"
```

Example 4: The intervals cannot have negative values.

```
"AOEXEC ALERT KEY(KEY4) TEXT('test alert') PRIORITY(INFO) ESCALATE(UP)" ,
"INTERVAL(, 10,-20) SSID(RE61)"
```

Examples of the PUBLISH Parameter

The following examples demonstrate the usage of the AOEXEC ALERT PUBLISH parameter.

Example 1: This example creates an ALERT and publishes it to all connected PATROL EM workstations, deleting any ALERTs already present with the same queue name and key.

```
"AOEXEC ALERT KEY(TESTKEY) TEXT('THIS IS A TEST') FUNCTION(ADD) PUBLISH(REPLACE)" ,
  "QUEUE(TEST AREA) SSID(RE61)"
```

Example 2: This example creates an ALERT but does not publish it to any connected MAINVIEW AutoOPERATOR Elan Workstation.

```
"AOEXEC ALERT KEY(TESTKEY) TEXT('DO NOT PUBLISH ME') FUNCTION(ADD) PUBLISH(NO)" ,
  "QUEUE(MAIN) SSID(RE61)"
```

AOEXEC MSG

This command logs a message in the BBI-SS PAS Journal log.

Command	Parameters
AOEXEC MSG	'Message text' SS | SSID(subsystem identifier) [TGTSS(target subsystem identifier)]

The following table describes the parameters.

Parameter	Function	Notes
Message text	Text of the message to issue.	Maximum length is 252 bytes.
SS | SSID	SS | SSID() specifies the subsystem identifier of a local subsystem.	Required keyword.
TGTSS	If the TGTSS() keyword is specified, the subsystem specified by the SS | SSID() keyword is considered a router and the actual function is executed on the subsystem specified by TGTSS(). If TGTSS() is not specified, the requested function is executed on the subsystem specified by the SS | SSID keyword.	Optional keyword. It must be in the same sysplex as the BBI-SS specified with the SSID() keyword, and both systems must have the same XCFGROUP specified in the BBPARM BBISSPxx.

Note: Specifying a null variable for Message text causes an error.

Return codes are listed in the following table.

Value	Description
-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
0	Command was executed successfully.
8	Supplied message text exceeds limit of 252 characters.
16	Invalid syntax used.
32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as the originating subsystem.
36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
40	Security definitions disallowed access to this function on the specified subsystem.
48	Incompatible versions of AOAnywhere exist.
52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Example

This example sends a message to the BBI-SS PAS monitoring the target named CICA. The message is logged on the remote Journal and no entry is made on the originating system's Journal.

```
“AOEXEC MSG 'MANUFACTURING DATABASE IS OFFLINE' SSID(RE61) TGTSS(CICA)”
```

AOEXEC NOTIFY

This command sends a request through AutoOPERATOR to issue a pager call using the MAINVIEW AutoOPERATOR Elan workstation component (if it is installed).

Command	Parameters
AOEXEC NOTIFY	NAME(Elan contact name) [INFO('Text')] SS | SSID(subsystem identifier) [TGTSS(target subsystem identifier)]

The following table describes the parameters.

Parameter	Function	Notes
NAME	The contact name defined to MAINVIEW AutoOPERATOR Elan workstation.	1-32 characters alphanumeric. MAINVIEW AutoOPERATOR Elan workstation equates this name to a telephone number to be dialed.
INFO	Any information to be passed and placed on the pager.	1-12 alphanumeric characters. Text must be included in quotation marks if it contains blanks.
SS | SSID	SS | SSID() specifies the subsystem identifier of a local subsystem.	Required keyword.
TGTSS	If the TGTSS() keyword is specified, the subsystem specified by the SS | SSID() keyword is considered a router and the actual function is executed on the subsystem specified by TGTSS(). If TGTSS() is not specified, the requested function is executed on the subsystem specified by the SS | SSID keyword.	Optional keyword. It must be in the same sysplex as the BBI-SS specified with the SSID() keyword, and both systems must have the same XCFGROUP specified in the BBPARM BBISSPxx.

Return codes are listed in the following table

Value	Description
-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
0	MAINVIEW AutoOPERATOR Elan workstation successfully passed the information.
8	The request timed out.
12	MAINVIEW AutoOPERATOR Elan workstation could not execute the request.
16	MAINVIEW AutoOPERATOR Elan workstation communications were not established.

Value	Description
32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as originating subsystem.
36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
40	Security definitions disallowed access to this function on the specified subsystem.
48	Incompatible versions of AOAnywhere exist.
52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Example

This command notifies the individual SYSPROG through MAINVIEW AutoOPERATOR Elan Workstation, passing the information SYSTEM to the pager.

```
"AOEXEC NOTIFY NAME(SYSPROG) INFO(SYSTEM) SSID(RE61)"
```

AOEXEC SELECT

This command invokes an EXEC or a program. This section also describes how to invoke programs written in other programming languages.

Command	Parameters
AOEXEC SELECT	EXEC(‘execname parm1...parm2...parm*n*’) [PRI(NORMAL|HIGH)] [WAIT(NO|YES)] SS | SSID(subsystem identifier) [TGTSS(target susbsystem identifier)] [VAR(var1....var2....var3...var*n*)]

The following table describes the parameters.

Parameter	Function	Notes
EXEC(‘execname and any parms’)	Name of EXEC to invoke. If there are parameters, the EXEC name and the parameters must be enclosed in quotation marks. If only the EXEC name is specified, do not use quotation marks.	Maximum length is 255 characters. Required parameter.
PRI	Execution priority of the EXEC to be invoked	Either NORMAL or HIGH. Applies only to EXEC keyword. It overrides AAOEXP00 parameters. PRI is valid with WAIT(YES) and WAIT(NO).
WAIT	Suspension criterion for invoking EXEC	Either YES or NO. WAIT(YES) causes the AOEXEC command to be suspended until the invoked EXEC in the BBI-SS PAS has completed. WAIT(NO) is returned as soon as a determination has been made whether the EXEC to be invoked actually exists. When VAR() is specified WAIT(YES) will be forced.
SS | SSID	SS | SSID() specifies the subsystem identifier of a local subsystem.	Required keyword.

Parameter	Function	Notes
TGTSS	If the TGTSS() keyword is specified, the subsystem specified by the SS | SSID() keyword is considered a router and the actual function is executed on the subsystem specified by TGTSS(). If TGTSS() is not specified, the requested function is executed on the subsystem specified by the SS | SSID keyword.	Optional keyword. It must be in the same sysplex as the BBI-SS specified with the SSID() keyword, and both systems must have the same XCFGROUP specified in the BBPARM BBISSPxx.
VAR	Specifies the names of any number of variables that will be exchanged with the LOCAL variable pool of the selected EXEC.	Before the target EXEC begins, the contents of these variables are placed as variables of the same name into the LOCAL pool. When the EXEC ends, the contents of these variables in the target EXEC's LOCAL pool are extracted again and placed as TSO variables of the EXEC of the invoking EXEC.

Return codes are listed in the following table and are set to the value specified in the IMFEXEC EXIT statement of the calling EXEC.

Value	Description
-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
0	Command was executed successfully.
8	EXEC specified but is not found in BBPROC.
16	Invalid syntax used.
32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as originating subsystem.
36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
40	Security definitions disallowed access to this function on the specified subsystem.
48	Incompatible versions of AOAnywhere exist.
52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Example

This example command invokes the EXEC CHKENQ on the remote SS SYSB, passing the parameter SYS2.PROD.XLIB.

```
"AOEXEC SELECT EXEC('CHKENQ SYS2.PROD.XLIB') SSID(RE61) TGTSS(SYSB)"
```

AOEXEC SYSINFO

This command searches the current MVS image for an AutoOPERATOR subsystem that runs AOAnywhere support. It returns information in variables regarding the success and failure of this search, as well as the XCF group name in which the targeted (or defaulted to) subsystem resides. Additionally, it returns the identifiers of all AutoOPERATOR subsystems that are connected to each other (in the sysplex) and it identifies those subsystems that have been designated as Alert Receivers.

This information can be used in subsequent requests against AOAnywhere, which require the presence of a SSID identifier.

The minimum required version level for an AutoOPERATOR subsystem to support AOAnywhere is 6.1.

This command has the following parameters.

Command	Parameters
AOEXEC SYSINFO	[SS | SSID()] [GROUP()]

The following table describes the parameters.

Parameter	Function	Notes
SS | SSID	SS | SSID() specifies the subsystem identifier from which system information is obtained. One to four alphanumeric characters. Optional.	This parameter should be used only when separate XCF groups will be used within a sysplex. An XCF group for a specific subsystem is specified on the XCFGROUP= parameter in BBPARM member BBISSP00. When this parameter is specified, only information for the subsystems connected to the same XCF group as the targeted subsystem is obtained. This parameter should not be used in conjunction with the GROUP() parameter. When neither SSID() nor GROUP() is specified, GROUP(BMCAB) is the default. The first subsystem on the current OS/390 image belonging to this group will be referenced to obtain information about all other AutoOPERATOR subsystems connected to each other through this XCF group.
GROUP	GROUP() specifies the XCF group from which information is obtained. One to eight alphanumeric characters in accordance with IBM XCF group names. Optional.	This parameter should be used only when separate XCF groups will be used within a sysplex. An XCF group for a specific subsystem is specified on the XCFGROUP= parameter in BBPARM member BBISSP00. When this parameter is specified, only information for the subsystems connected to the same XCF group as the targeted subsystem is obtained. At least one AutoOPERATOR subsystem that belongs to the specified XCF group should reside on the current OS/390 image. Otherwise this request will fail. XCFGROUP() or XCF() are valid aliases of this command. When neither SSID() nor GROUP() is specified, GROUP(BMCAB) is the default. The first subsystem on the current OS/390 image belonging to this group will be referenced to obtain information about all other AutoOPERATOR subsystems connected to each other through this XCF group.

The following table describes the variables that AOEXEC SYSINFO returns.

Variable Name	Description
SYSTEM	The name of the current OS/390 image (commonly referred to as the system name)
ALRT1 through ALRTx	A value of YES or NO. YES means this subsystem has been designated as an ALERT receiver by specifying ALRTRCVE=YES in BBPARM member BBISSP00. Otherwise the returned value is NO.
IMFXCFGP	The name of the default or target XCF group referred to by the command. If the SSID() parameter is specified, it contains the name of the XCF group of which the targeted subsystem is a member. If GROUP() was specified, the name is identical to the contents of this keyword.
LCNT	The number of lines returned.
SSID1 through SSIDx	An AutoOPERATOR subsystem (SSID) name that is supporting AOAnywhere where x is between 1 and the value contained in LCNT variable.
SYSN1 through SYSNx	The names of the MVS images that the relative SSID is active on where x is between 1 and the value contained in LCNT .
The SSID, SYSN and ALRT variables are returned in triplets. For example, SSID1, SYSN1 and ALRT1 are returned together; SSID2, SYSN2 and ALRT2 are returned together, and so on.	

Return codes are listed in the following table.

Value	Description
44	Processing was terminated in the middle of processing an AOEXEC SYSINFO. If more than one AutoOPERATOR PAS capable of processing a AOEXnEC SYSINFO command is active on the local system, it is possible that this situation is temporary and a subsequent execution of SYSINFO will be successful.
52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Example

```
/* REXX */
"AOEXEC SYSINFO"
if rc<> 0 then do
  say 'No active subsystems found'
  exit
end
do i=1 to lcnt
  if value('SYSN'i)=system then do
    myss=value('SSID'i)
    mysys=value('SYSN'i)
    leave
  end
end
"AOEXEC VGET VAR(QJNLSTA) SSID("myss")"
say 'Current journaling status on 'strip(mysys)' is 'qjnlsta
```

AOEXEC VDEL

This command deletes one or more variables from one of the AutoOPERATOR variable pools.

Command	Parameters
AOEXEC VDEL	[POOL(SHARED|PROFILE)] SS | SSID(subsystem identifier) [TGTSS(target susbsystem identifier)] [VAR(var1....var2....var3...var*n*)]

The following table describes the parameters.

Parameter	Function	Notes
POOL	The pool in which the designated variables reside	One of the following pools: • SHARED • PROFILE SHARED is the default.
SS | SSID	SS | SSID() specifies the subsystem identifier of a local subsystem.	Required keyword.
TGTSS	If the TGTSS() keyword is specified, the subsystem specified by the SS | SSID() keyword is considered a router and the actual function is executed on the subsystem specified by TGTSS(). If TGTSS() is not specified, the requested function is executed on the subsystem specified by the SS | SSID keyword.	Optional keyword. It must be in the same sysplex as the BBI-SS specified with the SSID() keyword, and both systems must have the same XCFGROUP specified in the BBPARM BBISSPxx.

Parameter	Function	Notes

VAR	The name of one or more variables	The maximum length of this parameter is 252 bytes. All variables in a pool can be deleted by using the identifier ALL instead of naming all variables individually. A variable cannot begin with a numeric nor can it contain special characters.
		
		An example of using a pattern is
		
		`AOEXEC VDEL VAR(CICS*) SSID(RE61)`
		
		The variable names can be generically expressed by using an asterisk. However, the VDEL command statement assumes the presence of an asterisk means the end of the string.
		
		`AOEXEC VDEL VAR(ABC*D) SSID(RE61)`
		
		is treated as if you coded
		
		`AOEXEC VDEL VAR(ABC*) SSID(RE61)`
		
		In addition, if you try to use an asterisk within a string of text, you will receive a return code for invalid syntax usage. For example, if you try to issue a pattern
		
		`AOEXEC VDEL VAR(CSM*MSG12) SSID(RE61)`
		
		you will receive a return code of IMFCC=16 (for invalid syntax usage).
		
		Variables beginning with the character Q are reserved for system variables and cannot be modified.

Note: This command does not affect variables that have already been retrieved from one of the pools.

Return codes are listed in the following table.

Value	Description
-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
0	Command was executed successfully.
8	Variable does not exist.
16	Invalid syntax used.
20	Severe error (internal) and pool was not found.

24	Variable name not specified.
32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as originating subsystem.
36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
40	Security definitions disallowed access to this function on the specified subsystem.
48	Incompatible versions of AOAnywhere exist.
52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Example

This example deletes all variables ending in the characters TEST from the shared variable pool. It uses the VLST command to retrieve all variable names.

```
"AOEXEC VLST VAR(*) POOL(SHARED) SSID(RE61)"
do i=1 to lcnt
  if length(value('line'.i))< 4 then iterate
  if right(value(value(line.i)), 4)='TEST' then
  "AOEXEC VDEL VAR("value(line.i)") POOL(SHARED) SSID(RE61)"
end
```

AOEXEC VGET

This command copies one or more variables from one of the AutoOPERATOR pools into the EXECs function pool.

Command	Parameters
AOEXEC VGET	[POOL(SHARED|PROFILE)] SS | SSID(subsystem identifier) [TGTSS(target susbsystem identifier)] VAR(var1....var2....var3...var*n*)

The following table describes the parameters.

Parameter	Function	Notes
POOL	The pool in which the designated variables reside	One of the following pools: • SHARED • PROFILE SHARED is the default.
SS | SSID	SS | SSID() specifies the subsystem identifier of a local subsystem.	Required keyword.
TGTSS	If the TGTSS() keyword is specified, the subsystem specified by the SS | SSID() keyword is considered a router and the actual function is executed on the subsystem specified by TGTSS(). If TGTSS() is not specified, the requested function is executed on the subsystem specified by the SS | SSID keyword.	Optional keyword. It must be in the same sysplex as the BBI-SS specified with the SSID() keyword, and both systems must have the same XCFGROUP specified in the BBPARM BBISSPxx.
VAR	The name of one or more variables	Each variable name can be up to 32 characters. The maximum length of the combined variable values is 252 bytes.

Return codes are listed in the following table.

Value	Description
-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
0	Command was executed successfully.
8	Variable does not exist.
12	Variable name not specified.
16	Invalid syntax used.
20	Severe error (internal) and pool was not found.

Value	Description
32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as originating subsystem.
36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
40	Security definitions disallowed access to this function on the specified subsystem.
48	Incompatible versions of AOAnywhere exist.
52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Example

This example displays the contents of the EXECs SHARED variable pool. It uses the VLST command to retrieve the names of all variables in that pool.

It then uses the VGET command to retrieve them one after the other and displays their contents.

```
"AOEXEC VLST VAR(*) POOL(SHARED) SSID(RE61)"
do i=1 to lcnt
 var=value(value(line)i)
 "AOEXEC VGET VAR("var") POOL(SHARED) SSID(RE61)"
 say var
end
```

AOEXEC VLST

This command lists variable names defined in the AutoOPERATOR pools. It returns those names in LOCAL variables LINE1 through LINEn and sets LCNT to the number of LINEs.

Command	Parameters
AOEXEC VLST	[POOL(SHARED|PROFILE)] SS | SSID(subsystem identifier) [TGTSS(target susbsystem identifier)] VAR(variable name)

The following table describes the parameters.

Parameter	Function	Notes
POOL	The pool in which the designated variables reside	One of the following pools: • SHARED • PROFILE SHARED is the default.
SS | SSID	SS | SSID() specifies the subsystem identifier of a local SS.	Required keyword

TGTSS	If the TGTSS() keyword is specified, the subsystem specified by the SS | SSID() keyword is considered a router and the actual function is executed on the subsystem specified by TGTSS(). If TGTSS() is not specified, this is the subsystem where the requested function is executed.	Optional keyword. It must be in the same sysplex as the BBI-SS specified with the SSID() keyword, and both systems must have the same XCFGROUP specified in the BBPARM BBISSPxx.
VAR	The name of one variable	Required parameter. Only one variable can be specified and the name must be enclosed in parentheses. The variable name can be 1-30 characters alphanumeric conforming to TSO coding conventions. The variable name can be a pattern `(A+B*)` where the following wildcards are supported: + **(plus sign)** Matches any one character. *** (asterisk)** Matches zero to any number of characters.

Return codes are listed in the following table.

Value	Description
-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
0	Command was executed successfully.
12	Variable pool is not available.
16	Invalid syntax used.
20	Variable name not specified.
32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as originating subsystem.
36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
40	Security definitions disallowed access to this function on the specified subsystem.

48	Incompatible versions of AOAnywhere exist.
52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Example

The following EXEC uses the AOEXEC VLST command to retrieve all the variables that begin with RETRY and then reports the number of retries. Variables LINE1 through LINExx (where xx is IMFNOL) will contain the number of found variables.

```
/* REXX */
"AOEXEC VLST VAR(RETRY*) POOL(SHARED) SSID(RE61)"
if rc = 8 then exit
do i=1 to lcnt
  contents=value('LINE'I)
  "AOEXEC VGET VAR("contents") SSID(RE61)"
  contents=value(contents)
  count=left(contents,6)
  nod=substr(contents,7)
  say 'Terminal : 'nod' Retries: 'count
END
```

AOEXEC VPUT

This command copies one or more variables from the EXECs function pool into one of the AutoOPERATOR pools.

Command	Parameters
AOEXEC VPUT	[POOL(SHARED|PROFILE)] SS | SSID(subsystem identifier) [TGTSS(target subsystem identifier)] VAR(var1....var2....var3...var*n*)

The following table describes the parameters.

Parameter	Function	Notes
POOL	The pool in which the designated variables reside	One of the following pools: • SHARED • PROFILE SHARED is the default.
SS | SSID	SS | SSID() specifies the subsystem identifier of a local SS.	Required keyword.
TGTSS	If the TGTSS() keyword is specified, the subsystem specified by the SS | SSID() keyword is considered a router and the actual function is executed on the subsystem specified by TGTSS(). If TGTSS() is not specified, this is the subsystem where the requested function is executed.	Optional keyword. It must be in the same sysplex as the BBI-SS specified with the SSID() keyword, and both systems must have the same XCFGROUP specified in the BBPARM BBISSPxx.
VAR	The name of one or more variables	Each variable name can be up to 32 characters. The maximum length of the combined variable values is 252 bytes. Variables beginning with the character Q are reserved for system variables and should not be modified.

Return codes are listed in the following table.

Value	Description
-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
0	Command was executed successfully.
4	Variable did not previously exist in the designated pool.
12	Q-type variable was specified and cannot be copied with VPUT.
16	Invalid syntax used.

Value	Description
20	Variable name is invalid.
24	Variable name was not specified.
32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as originating subsystem.
36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
40	Security definitions disallowed access to this function on the specified subsystem.
48	Incompatible versions of AOAnywhere exist.
52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Examples

This section contains examples using the AOEXEC VPUT command statement. A brief discussion follows each example.

Example 1.

```
"AOEXEC VPUT VAR(ABENDS ABENDCOUNT ABENDREASON) POOL(SHARED) SSID(RE61)"
```

This example command saves the current value of `ABENDS`, `ABENDCOUNT`, and `ABENDREASON` in the SHARED pool.

Example 2.

```
"AOEXEC VPUT VAR(ABENDS) POOL(PROFILE) SSID(RE61)"
```

This example command saves the current value of `ABENDS` in the PROFILE pool.

AOEXEC VDELL

This command deletes one or more long variables from one of the AutoOPERATOR variable pools.

Note: This variable operation only supports a subset of the functions available for the short variables. It ONLY affects and searches for long variables. If a short variable (created with VPUT instead of VPUTL) with the specified name exists, it is ignored.

Command	Parameters
AOEXEC VDELL	[POOL(SHARED|PROFILE)] VAR(var1....var2....var3...var*n*) SS | SSID(subsystem identifier) [TGTSS(target subsystem identifier)]

The following table describes the parameters.

Parameter	Function	Notes
POOL	The pool in which the designated variables reside	One of the following pools: • SHARED • PROFILE SHARED is the default.
VAR	The name of one or more variables	Required parameter. Each variable name can be up to 32 characters. Maximum parameter length is 252. Variables beginning with the character Q are reserved for system variables and cannot be modified.
SS | SSID	SS | SSID() specifies the subsystem identifier of a local SS.	Required keyword.
TGTSS	If the TGTSS() keyword is specified, the subsystem specified by the SS | SSID() keyword is considered a router and the actual function is executed on the subsystem specified by TGTSS(). If TGTSS() is not specified, this is the subsystem where the requested function is executed.	Optional keyword. It must be in the same sysplex as the BBI-SS specified with the SSID() keyword, and both systems must have the same XCFGROUP specified in the BBPARM BBISSPxx.

Note: This command does not affect variables that have already been retrieved from one of the pools.

Return codes are listed in the following table.

Value	Description
-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.

0	The variable existed in the target pool and has been deleted.
8	No long variable with this name has been found in the target pool.
12	An attempt to delete a read-only variable (for example, Q-type variable was specified which cannot be deleted with VDELL).
16	Invalid syntax used.
24	Variable name not specified.
32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as originating subsystem.
36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
40	Security definitions disallowed access to this function on the specified subsystem.
48	Incompatible versions of AOAnywhere exist.
52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Example

The PROFILE pool is searched for a long variable with the name of X. If found, it is deleted.

```
"AOEXEC VDELL VAR(X) POOL(PROFILE) SSID(RE61)"
```

AOEXEC VGETL

This command copies one or more long variables from one of the AutoOPERATOR pools into the TSO pool.

Note: This variable operation supports only a subset of the functions available for the short variables. It ONLY affects and searches for long variables. If a short variable (created with VPUT instead of VPUTL) with the specified name exists, it is ignored.

Command	Parameters
AOEXEC VGETL	[POOL(SHARED|PROFILE)] VAR(var1....var2....var3...var*n*) SS | SSID(subsystem identifier) [TGTSS(target subsystem identifier)]

The following table describes the parameters.

Parameter	Function	Notes
POOL	The pool in which the designated variables reside	One of the following pools: • SHARED • PROFILE SHARED is the default.
VAR	The name of one or more variables	Required parameter. Each variable name can be up to 30 characters.
SS | SSID	SS | SSID() specifies the subsystem identifier of a local SS.	Required keyword.
TGTSS	If the TGTSS() keyword is specified, the subsystem specified by the SS | SSID() keyword is considered a router and the actual function is executed on the subsystem specified by TGTSS(). If TGTSS() is not specified, this is the subsystem where the requested function is executed.	Optional keyword. It must be in the same sysplex as the BBI-SS specified with the SSID() keyword, and both systems must have the same XCFGROUP specified in the BBPARM BBISSPxx.

Return codes are listed in the following table.

Value	Description
-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
0	The variable existed in the target pool and has been retrieved.
12	Variable name not specified.
16	Invalid syntax used.

Value	Description
32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as originating subsystem.
36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
40	Security definitions disallowed access to this function on the specified subsystem.
48	Incompatible versions of AOAnywhere.
52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Examples

The PROFILE pool is searched for a long variable with the name of X. If found, it is placed into the TSO pool and assigned to the variable Y.

```
"AOEXEC VGETL VAR(X) POOL(PROFILE) SSID(RE61)"
Y=X
```

AOEXEC VLSTL

This command retrieves a long variable from the specified pool and places it into the TSO pool.

Note: This variable operation supports only a subset of the functions available for the short variables. It ONLY affects and searches for long variables. If a short variable (created with VPUT instead of VPUTL) with the specified name exists, it is ignored.

Command	Parameters
AOEXEC VLSTL	[POOL(SHARED|PROFILE)] VAR(var) SS | SSID(subsystem identifier) [TGTSS(target subsystem identifier)]

The following table describes the parameters.

Parameter	Function	Notes
POOL	The pool in which the designated variables reside	One of the following pools: • SHARED • PROFILE SHARED is the default.
VAR	The name of one variable	Required parameter. Only one variable can specified and the name must be enclosed in parentheses. Each variable name can be up to 30 characters. The variable name can be a pattern `(A+B*)` where the following wildcards are supported: + **(plus sign)** Matches any one character. *** (asterisk)** Matches zero to any number of characters.

SS | SSID	SS | SSID() specifies the subsystem identifier of a local SS.	Required keyword.
TGTSS	If the TGTSS() keyword is specified, the subsystem specified by the SS | SSID() keyword is considered a router and the actual function is executed on the subsystem specified by TGTSS(). If TGTSS() is not specified, this is the subsystem where the requested function is executed.	Optional keyword. It must be in the same sysplex as the BBI-SS specified with the SSID() keyword, and both systems must have the same XCFGROUP specified in the BBPARM BBISSPxx.

Return codes are listed in the following table..

Value	Description
-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
0	At least one variable has been found.
16	Invalid syntax used.
20	Variable name not specified.
32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword. The target subsystem is most likely not active or not in the same sysplex as originating subsystem.
36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
40	Security definitions disallowed access to this function on the specified subsystem.
48	Incompatible versions of AOAnywhere.
52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Example

This EXEC lists all long variables in the SHARED pool and writes their names to the terminal.

```
/* REXX */
"AOEXEC VLSTL VAR(*) POOL(SHARED) SSID(RE61)"
say lcnt
do i=1 to lcnt
  name = value('line'i)
  say name
end
```

AOEXEC VPUTL

This command creates a or sets a long variable from a variable in the TSO pool.

Note: This variable operation supports only a subset of the functions available for the short variables. For example, no target system functionality is provided. It ONLY affects and searches for long variables. If a short variable (created with VPUT instead of VPUTL) with the specified name exists, it is ignored.

Command	Parameters
AOEXEC VPUTL	[POOL(SHARED|PROFILE)] VAR(var1....var2....var3...var*n*) SS | SSID(subsystem identifier) [TGTSS(target subsystem identifier)]

The following table describes the parameters.

Parameter	Function	Notes
POOL	The pool in which the designated variables reside	One of the following pools: • SHARED • PROFILE SHARED is the default.
VAR	The name of one or more variables	Required parameter. Each variable name can be up to 30 characters. Variables beginning with the character Q are reserved for system variables and cannot be modified.
SS | SSID	SS | SSID() specifies the subsystem identifier of a local SS.	Required keyword.
TGTSS	If the TGTSS() keyword is specified, the subsystem specified by the SS | SSID() keyword is considered a router and the actual function is executed on the subsystem specified by TGTSS(). If TGTSS() is not specified, this is the subsystem where the requested function is executed.	Optional keyword. It must be in the same sysplex as the BBI-SS specified with the SSID() keyword, and both systems must have the same XCFGROUP specified in the BBPARM BBISSPxx.

Return codes are listed in the following table.

Value	Description
-1	When the TGTSS() keyword is used, specifies that either the request timed out or the target system was shut down in the middle of a request.
0	The variable existed in the target pool and has been overwritten.
4	The variable did not exist in the pool and has been created.
8	An error occurred during operation. Possible out-of-space condition for the PROFILE pool.
12	An attempt was made to set a read-only variable (for example, Q-type variable was specified which cannot be set with VPUT).
16	Invalid syntax used.
20	Variable pool not found. BIVARS not allocated.
24	Variable name not specified.
32	No XCF connection exists between the subsystem specified with the SS | SSID() keyword and the subsystem specified using the TGTSS() keyword.cannot be found The target subsystem is most likely not active or not in the same sysplex as originating subsystem.
36	The subsystem specified using the SS | SSID() keyword cannot be found on the local system.
40	Security definitions disallowed access to this function on the specified subsystem.
48	Incompatible versions of AOAnywhere.
52	An attempt was made to execute this function under NetView without a valid Access/NetView product key.

Examples

This example saves the variable A to the SHARED pool. Note that the variable can be shorter than 255 characters.

```
A='This is a test'
"AOEXEC VPUTL VAR(A) POOL(SHARED) SSID(RE61)"
```

Chapter 10. Accessing Array Data with AutoOPERATOR EXECs

This chapter describes how to use IMFEXEC ARRAY commands to access data collected in arrays.

Overview

This document outlines IMFEXEC ARRAY|ARY commands that enable you to access data from two-dimensional variable arrays. Arrays bear some resemblance to ISPF tables in the way they can be scanned, sorted, positioned and are backed by disk space.

When Are Arrays Useful

An EXEC often has to deal with many instances of the same data type for example: a number of unit addresses, TSO/E user names, job names, and so on. Often these data types are part of a record type. In addition to a job name, a job also has a jobstep name, a start time, elapsed CPU time, EXCPs and any amount of additional information.

These data fields can be manipulated using REXX stem variables but only for a single column or field in a record. TSO/E CLIST EXECs cannot handle this type of data at all. When dealing with multiple fields, you might use several REXX stem variables with the same index but many inefficient operations can result when swapping records or assigning them to a third record. Instead of referring to a single record, REXX stem variables force you to deal with fields only, never considering them as related items.

Furthermore, scanning these records for particular contents or sorting and creating specific subsets of information becomes cumbersome and resource intensive.

This is where arrays come in: arrays represent data in row-column format where data items are kept together in rows or records. Instead of manipulating this data manually, certain operations may be performed against an array as an entirety, such as sorting it based upon the contents of a column.

To process an array you create a reference to a specific row (also called a record) and retrieve the entire row into REXX variables. This operation potentially sets a great number of variables all at once. A row is always treated as one unit and the individual fields will never lose synchronization (which might occur when using individual REXX stem variables).

Other advantages to arrays include

- Rows can be filtered so that only those rows whose columns meet certain criteria are visible
- Rows can be sorted with one command
- Arrays can be shared among multiple EXECs and saved to permanent storage (DASD)

As a debugging aid, a sample Exec (DUMPARY) that writes the contents of an Array to the BBI journal is included in the BBSAMP library. You can invoke this EXEC from a BBI command line by passing to it, the name of the Array and the number of rows and columns to be displayed, for example:

```
%DUMPARY ARRAY(array) ROWS(50) COLS(10)
```

where array is the name of an array saved on disk, and 50 rows with 10 columns in each row will be displayed. Additionally, you can invoke DUMPARY from within your own EXEC by specifying one of the following commands:

```
IMFEXEC SELECT EXEC(DUMPARY ARRAY(array) ROWS(n) COLS(n) CON(N)) WAIT(YES)
or
call DUMPARY 'ARRAY(array) ROWS(n) COLS(n) CON(N)'
```

where array is the name of an array currently accessed by your EXEC. ARRAY is the only required parameter. All parameters can be abbreviated for convenience. The parameter CON(N) is used when your EXEC already has a connection to the array. For more information about abbreviations and examples of how to invoke the DUMPARY EXEC, invoke it from the BBI journal command line, passing to it, the text '*help*'. Then read the output in the BBI journal. For example, specify:

```
%DUMPARY HELP
```

Note: If you do not specify a value for `ROWS()` or `COLS()`, the entire array will be written to the journal. Be sure that either the BBSAMP library is in your SYSPROC concatenation or copy the DUMPARY EXEC from BBSAMP to another library concatenated to SYSPROC.

IMFEXEC ARRAY Commands

The following table lists the IMFEXEC ARRAY commands you can use to access information in arrays and the page where you can find more information.

Command	Page	Function
CONNECT	193	Establishes a logical connection between one or more EXECs and an array.
CREATE	195	Defines a new array by providing definitions of its logical characteristics.
DELETE	197	Deletes a row from an array.
DISC	198	Terminates a logical connection between one or more EXECs and an array.
FIND	200	Locates a particular row conforming to a set of criteria.
GET	202	Transfers the current array row into local variables.
INFO	203	Provides information about an array.
INSERT	205	Inserts a new row into an array.
LIST	206	Provides information about saved or disconnected arrays (when kept).
PUT	207	Sets the current array row from local variables.
SAVE	208	Checkpoints the contents of an array to disk.
SET	209	Transfers an array into REXX TSO/E variables.
SETVIEW	210	Limits array access to rows matching certain criteria.
SORT	212	Sorts an array according to one or more criteria.

General Coding Conventions

The following sections briefly describe the coding conventions for using the IMFEXEC ARRAY | ARY command statements.

Note: Every command described in this chapter is prefixed by the literal ARRAY | ARY to avoid naming conflicts with existing IMFEXEC constructs. ARY is a valid abbreviation.

For example:

```
IMFEXEC ARRAY|ARY command [parameters]
```

Using Variable Names

Variable names are limited to 31 characters in length. The first character of the variable must be alphanumeric or one of the following special characters:

- $
- @
- #

Reading Condition Codes

Every command returns a condition code in the variable IMFCC in the TSO/E pool. Refer to Chapter 4, “Using Variables in REXX EXECs” on page 49 for more information about pools.

Each IMFEXEC command statement description includes a table describing the parameters for the command. The table uses the following format:

Parameter	Function	Notes
1	**2**	**3**

The numbers in this table correspond to the following descriptions:

1 A short parameter identifier. If the parameter has uppercase letters, this identifier must be coded exactly as shown.

If parts of the identifier are shown in **bold**, this parameter can be abbreviated, using the bold letters.

Positional parameters are not associated with a specific identifier. In these cases, this column contains an alias that describes the parameter.

2 The function of the parameter.

3 Notes about the parameter. Typically, these notes describe any length, value, range, or string limitations.

ARRAY CONNECT

This command establishes a logical connection between one or more EXECs and an array.

Command	Parameters
ARRAY|ARY CONNECT	NAME [ACCESS(UPDATE|READ)] [TOKEN()] [MSG|NOMSG]

The following table describes the parameters.

Parameter	Function	Notes
NAME	The name of the array as established during array creation	1-31 characters alphanumeric This parameter is required.
ACCESS	Array access definition	UPDATE is the default value. Multiple read accesses by separate threads to an array are possible. However, UPDATE requires exclusive access.
TOKEN	Array token returned by DISC KEEP	When not specified, the array is retrieved from DASD. When specified, only disconnected arrays are eligible.
Message option	Controls the writing of exception messages to the journal	One of the following values: **MSG** Exception messages are written to the journal. **NOMSG** No exception messages are written to the journal (default).

When retrieving an array from disc, the current position is at the very beginning of the array. Neither a View nor a Sort specification will exist. When reconnecting to a kept array position, Sort and View criteria will be exactly as left off.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully.
8	Array not found or error reading from disc / cannot create temp copy.
16	Syntax error.

Example

The EXEC attempts to establish a connection to the array named DASDSTATS that it assumes has been disconnected. If this attempt fails, a disc copy is loaded.

```
"IMFEXEC ARRAY CONNECT DASDSTATS TOKEN("arytoken")"
if imfcc <> 0 then "IMFEXEC ARRAY CONNECT DASDSTATS READ"
```

Note: After invoking the ARRAY CONNECT command, you can call the debugging EXEC, DUMPARY, that was first described in the section entitled “When Are Arrays Useful” on page 189. By adding one of the following statements following the ARRAY CONNECT command,

```
IMFEXEC SELECT EXEC(DUMPARY ARRAY(array) ROWS(n) COLS(n) CON(N)) WAIT(YES)
```

or

```
CALL DUMPARY 'ARRAY(array) ROWS(n) COLS(n) CON(N)'
```

where array is the name of the array returned by ARRAY CONNECT to your EXEC, you can write the contents of the array to the BBI journal.

Note that if you do not specify a value for ROWS() or COLS(), the entire array will be written to the journal.

ARRAY CREATE

This command defines a new array to AutoOPERATOR.

Command	Parameters
ARRAY|ARY CREATE	NAME STEM(stem name) INITIAL() [INC()]

The following table describes the parameters.

Parameter	Function	Notes
NAME	The name of the new array to define	1-31 characters alphanumeric
STEM	Variable root name of a set of variables containing the array definition	The format is identical to the format above. Under REXX, true stem variables will be referenced whereas under TSO/E a numeric is appended to the name. The low index is assumed to be 1. The definition continues until either a null or undefined variable is encountered.
INITIAL	Initial size in rows of the array	1-32767 numeric
INC	Increment to be used when extending the array	1-32767 numeric

After successful execution of the command the array will be in UPDATE access. It is possible to redefine arrays that currently exist and to overwrite them when saving. The format of the array definition in the indicated variables is as follows:

- Column name (1-255 chars, TSO/E conforming)
- Column width (1-32767, numeric)
- Format (must be C currently)
- User data pertaining to this field (1-32767 chars, no restrictions, optional)

Individual fields are separated by one or more spaces.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully.
8	Invalid or incomplete array definition.
16	Syntax error.

Example

This EXEC defines a new array with 3 columns unit: VOL and STAT. It does so by defining 3 variables with the contents of the definition. Although not absolutely necessary in this example, be sure the succeeding variable is set to null and the definition processor invoked.

The array definition is then saved.

```
array.1='UNIT 3 C'
array.2='VOL 6 C'
array.3='STAT 8 C'
array.4=''
"IMFEXEC ARRAY CREATE DASDSTATS STEM(ARRAY) INITIAL(500) INC(50)"
"IMFEXEC ARRAY DISC DASDSTATS SAVE"
```

ARRAY DELETE

This command deletes the current row from the array.

Command	Parameters
ARRAY|ARY DELETE	NAME

The following table describes the parameters.

Parameter	Function	Notes
NAME	The name of the array as established during array creation	1- to 31-characters alphanumeric.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
4	Array is empty
8	Array is not found
12	Array is not in UPDATE access
16	Syntax error

Example

This EXEC deletes all rows in an array beginning with those rows where variable VOL is greater than or equal to the string BAB.

```
"IMFEXEC ARRAY CONNECT DASDSTATS"
"IMFEXEC ARRAY FIND DASDSTATS CRITERIA('VOL,,,>=, ''BAB''') ROW(1)"
do while imfcc=0
  "IMFEXEC ARRAY DELETE DASDSTATS"
  "IMFEXEC ARRAY FIND DASDSTATS CRITERIA('VOL,,,>=, ''BAB''') ROW(1)"
end
```

ARRAY DISC

This command terminates a logical connection with an array.

Command	Parameters
ARRAY|ARY DISC	NAME [ACTION(SAVE|NOSAVE|DELETE|KEEP)]

The following table describes the parameters.

Parameter	Function	Notes
NAME	The name of the array as established during array creation	1- to 31-characters alphanumeric.
ACTION	Action to take upon termination	One of the following values: **SAVE** Saves all updates since the last save to disk and saves the cursor position **NOSAVE** Discards all changes since last save **DELETE** Discards all changes and removes array definition **KEEP** Retain array as-is in memory for future reference. See the following table for more information.

The following table describes the TSO/E variables returned from DISC (when KEEP is specified as the ACTION).

TSO/E Variables Returned from DISC			
NAME	**Contents**	**Length/Format**	**Notes**
ARYTOKEN	Token to be used to reconnect to the array	15/Character	Properties determined by internal design

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully.
8	Failure to save array not found (never connected or created).
16	Syntax error.

Example

This EXEC attempts to establish a connection with an array named DASDSTATS. If the array cannot be found it terminates with a message and a return code of 8. Otherwise it reads the first row from the array and then disconnects from it with the default action setting of NOSAVE.

```
"IMFEXEC ARRAY CONNECT DASDSTATS"
if imfcc <> 0 then do
  "IMFEXEC MSG '***** FATAL ERROR *****'"
  "IMFEXEC EXIT CODE(8)"
  exit
end
"IMFEXEC ARRAY FIND DASDSTATS ROW(1)"
"IMFEXEC ARRAY GET DASDSTATS"
"IMFEXEC ARRAY DISC DASDSTATS"
```

ARRAY FIND

This command positions an array's CURRENT row pointer at the first row meeting specified criteria.

Command	Parameters
ARRAY|ARY FIND	NAME [ROW] [CRITERIA]

The following table describes the parameters

Parameter	Function	Notes
NAME	The name of the array as established during array creation	1- to 31-characters alphanumeric.
ROW	Starting row for scan	The default is the current row Numeric, the first element of an array is indexed by 1.
CRITERIA	Criteria to which the row must conform	See the comments below the table.

Any number of criteria may be specified, connected by the Boolean operators AND and OR. Entries may be of one of the following two formats:

- Boolean operator (AND, OR), except for first criterion.
- Column name, 1-255 characters
- Starting position, default is 1
- Length used for comparison, default width of column+1 - starting position
- Comparison operator, one of the following: LT, LE, EQ, GE, GT, NE, <, < =, =, > =, >, < >, ^ =

Or

- Boolean operator (AND, OR), except for first criterion
- Column name, 1-255 characters
- Starting position, default is 1
- Length used for comparison, default width of column+1 - starting position
- Comparison operator, one of the following: LT, LE, EQ, GE, GT, NE, <, <=, =, >=, >, < >, ^=
- Literals are enclosed in quotation marks

The following rules also apply:

- When both fields contain numerics (except leading and trailing blanks), a numerical comparison is performed. Both numbers will always be treated as unsigned integers.
- When a comparison with a literal is requested, a pattern comparison is performed (for example, wildcards such as * and + may be used).

- When two columns with different lengths are compared, a comparison with the length of the shorter of the two is done.

Condition codes are listed in the following table.

Value	Description
0	Command was successfully executed.
4	Criteria parsing error.
8	Array not found.
12	Row specification past array extension or 0.
16	Syntax error.

Example

This EXEC connects to the array user ID that contains information about user IDs and accounts associated with them. It then finds all user IDs belonging to account 3911 and prints them in the BBI-SS log.

```
"IMFEXEC ARRAY CONNECT USERID"
"IMFEXEC ARRAY FIND USERID CRITERIA('ACCT,,,=,''3911''') ROW(1)"
do while imfcc=0
  "IMFEXEC ARRAY INFO USERID"
  "IMFEXEC ARRAY GET USERID"
  "IMFEXEC MSG "userid
"IMFEXEC ARRAY FIND USERID CRITERIA('ACCT,,,=,''3911''') ROW("arypos+1")"
end
```

ARRAY GET

This command identifies the view to be used for accessing data.

Command	Parameters
ARRAY|ARY GET	NAME [TRIM|NOTRIM] [SKIP|NOSKIP]

The following table describes the parameters.

Parameter	Function	Notes
NAME	The name of the array as established during array creation	1- to 31-characters alphanumeric.
TRIM	Defines whether leading and trailing blanks are removed (trimmed)	The default is TRIM which means blanks are removed.
SKIP	Advances the current row pointer by 1 after retrieving the contents of the row	Possible values are SKIP and NOSKIP. SKIP is the default.

Condition codes are listed in the following table.

Value	Description
0	Command was successfully executed.
4	Array is empty / no matching rows for SETVIEW.
8	Array not found.
16	Syntax error.
20	If SKIP was specified (or defaulted to) and the last row of the table was read, a return code of 20 will be returned.

Example

This EXEC locates the row containing the definition for volume BAB301 in the array named DASDSTATS. It then retrieves the current status of this volume and writes it to the log.

```
"IMFEXEC ARRAY CONNECT DASDSTATS"
"IMFEXEC ARRAY FIND DASDSTATS ROW(1) CRITERIA('VOL,,,=,''BAB301''')"
"IMFEXEC ARRAY GET DASDSTATS"
"IMFEXEC MSG VOL BAB301 Status: "stat
```

ARRAY INFO

This command identifies the view to be used for accessing data.

Command	Parameters
ARRAY|ARY INFO	NAME

The following table describes the parameters.

Parameter	Function	Notes
NAME	The name of the array as established during array creation	1- to 31-characters alphanumeric.

The following table describes the TSO/E variables returned from INFO.

TSO/E Variables Returned from INFO			
NAME	**Contents**	**Length/Format**	**Example**
ARYROWS	Number of active rows in the array	0-32767 numeric	
ARYFRAME	Number of total rows currently allocated for the array	1-32767 numeric	Corresponds to the initial() specification during array creation but may change as the array is extended
ARYINC	Increment used when extending the array	1-32767 numeric	Corresponds to the inc() specification during array creation
ARYSTOR	Total number of bytes occupied by the array itself and all associated control blocks	numeric	This includes the array itself, the array descriptor block, lookaside tables as well as sort and filter descriptor blocks.
ARYLROWS	Number of rows matching the current view	0-32767 numeric	
ARYCOLS	Number of columns of the array	1-32767 numeric	
ARYPOS	Current position within the array	1-32767 numeric	
ARYSORT	Specifies whether sort criteria have been attached to the array	YES or NO	

TSO/E Variables Returned from INFO			
ARYFILTER	Specifies whether a view has been attached to the array	YES or NO	
ARYCOLN.n	The name of all columns of the array	1-255, character	ARYCOLN.1, ARYCOLN.2 etc.
ARYCOLW.n	The width of the indicated column	1-32767 numeric	ARYCOLW.1, ARYCOLW.2 etc.

Condition codes are listed in the following table.

Value	Description
0	Command was successfully executed.
8	Array not found.
16	Syntax error.

Example

This EXEC connects to the array named DASDSTAT and establishes a view of the array which makes only those rows eligible where column STAT has the contents of ACTIVE. It then sorts the resulting array by the contents of column VOL and produces a list of all matching rows.

```
"IMFEXEC ARRAY CONNECT DASDSTAT"
"IMFEXEC ARRAY SETVIEW DASDSTAT CRITERIA('STAT,,,=,''ACTIVE''')"
"IMFEXEC ARRAY SORT DASDSTAT CRITERIA('VOL,,,A')"
"IMFEXEC ARRAY INFO DASDSTAT"
"IMFEXEC MSG The following volumes are active:"
do I =1 to arylrows
  "IMFEXEC ARRAY GET DASDSTAT"
  "IMFEXEC ARRAY MSG "vol
end
```

ARRAY INSERT

This command identifies the view to be used for accessing data.

Command	Parameters
ARRAY|ARY INSERT	NAME [POSITION(HERE|FIRST|LAST)]

The following table describes the parameters.

Parameter	Function	Notes
NAME	The name of the array as established during array creation	1- to 31-characters alphanumeric.
POSITION	Position in the array where row will be inserted	Could be one of the following values: **HERE** At the current position **FIRST** At the top of the array (element 1) **LAST** As the last element of the element If the array is ordered, the current position will always be determined by the sort criteria and field contents and this specification will be ignored.

Condition codes are listed in the following table.

Value	Description
0	Command was successfully executed.
8	Array not found.
12	Array not in UPDATE access.
16	Syntax error.

Example

This EXEC inserts a new row into the array referenced by the token contained in the variable DASDSTATS. It then sets the value of this row and checkpoints the contents of the array to permanent storage.

```
unit=3E0
vol=BAB301
stat=ACTIVE
"IMFEXEC ARRAY INSERT DASDSTATS"
"IMFEXEC ARRAY SAVE DASDSTATS"
```

ARRAY LIST

This command provides information about arrays in this BBI-SS PAS.

Command	Parameters
ARRAY|ARY LIST	[KEPT]

The following table describes the parameters.

Parameter	Function	Notes
KEPT	Provide information about disconnected arrays in storage (DISC with KEEP)	

The following table describes the TSO/E variables returned from LIST.

TSO/E Variables Returned from LIST			
NAME	**Contents**	**Length/Format**	**Example**
ARYNAMEXX	Name of the array	1-31 characters	
ARYTOKNxx	Token of the array if disconnected	15 characters	
ARYCOUNT	Count of arrays found		

Condition codes are listed in the following table.

Value	Description
0	Command was successfully executed.
16	Syntax error.

Example

This EXEC locates all disconnected arrays. It subsequently deletes all of them.

```
"IMFEXEC ARRAY LIST KEPT"
do j=1 to arycount
  "IMFEXEC ARRAY CONNECT "aryname.j" TOKEN("arytokn.j")"
  "IMFEXEC ARRAY DISC "aryname.j" NOSAVE"
end
```

ARRAY PUT

This command sets values of the current row of an array.

Command	Parameters
ARRAY|ARY PUT	NAME

The following table describes the parameters.

Parameter	Function	Notes
NAME	The name of the array as established during array creation	1- to 31-characters alphanumeric.

Condition codes are listed in the following table.

Value	Description
0	Command was successfully executed
8	Invalid array token
12	Array not in UPDATE access
16	Syntax error

Example

This EXEC locates the row containing the definition for volume BAB301 in the array DASDSTATS. It then sets the current status of this volume.

```
"IMFEXEC ARRAY CONNECT DASDSTATS UPDATE"
"IMFEXEC ARRAY FIND DASDSTATS ROW(1) CRITERIA('VOL,,,=,''BAB301''')"
"IMFEXEC ARRAY GET DASDSTATS"
stat=ACTIVE
"IMFEXEC ARRAY PUT DASDSTATS"
"IMFEXEC ARRAY DISC DASDSTATS SAVE"
```

ARRAY SAVE

This command checkpoints an AutoOPERATOR array. An array that was connected with the PAGE keyword cannot be saved. Use DISC/CONNECT for these arrays instead.

Command	Parameters
ARRAY|ARY SAVE	NAME

The following table describes the parameters.

Parameter	Function	Notes
NAME	The name of the array as established during array creation	1- to 31-characters alphanumeric.

Condition codes are listed in the following table.

Value	Description
0	Command was successfully executed.
8	Array not found or I/O error writing to disk.
16	Syntax error.

Example

This EXEC inserts a new row into the array referenced by the token contained in the variable DASDSTATS. It then sets the value of this row and checkpoints the contents of the array to permanent storage.

```
unit=3E0
vol=BAB301
stat=ACTIVE
"IMFEXEC ARRAY INSERT DASDSTATS"
"IMFEXEC ARRAY SAVE DASDSTATS"
```

ARRAY SET

This command transfers the entire contents of an array into REXX variables.

Command	Parameters
ARRAY|ARY SET	NAME [TRIM|NOTRIM]

The following table describes the parameters.

Parameter	Function	Notes
NAME	The name of the array as established during array creation	1- to 31-characters alphanumeric.
TRIM	Defines whether leading and trailing blanks are removed (trimmed)	The default is NOTRIM.

This command takes all rows and columns and creates REXX variables from them. Each variable name is identical to the column that it was derived from. A period '.' is then appended (effectively turning it into a stem variable) and then a counter is added for the row that it was copied from.

For example: For a column of UNIT, TSO/E variables of the name UNIT.1 to UNIT.xx are created.

This command functions properly only on column names that do not exceed 26 characters in length (since it appends .(_)xxxxx to the variable name). If column names exceeding 26 characters in length are found, a return code of 8 is returned.

Condition codes are listed in the following table.

Value	Description
0	Command was successfully executed.
4	Array is empty.
8	Array not found or column name wider than 26 characters was found.
16	Syntax error.

Example

This EXEC connects to the array DASDSTAT and sets all columns and rows to their respective REXX variables. It then writes a message to the log designating the contents of the 'unit' column of the first row.

```
"IMFEXEC ARRAY CONNECT DASDSTAT"
"IMFEXEC ARRAY SET DASDSTAT"
"IMFEXEC ARRAY DISC DASDSTAT NOSAVE"
"IMFEXEC MSG "unit.1
```

ARRAY SETVIEW

This command limits access to an array to those rows matching certain criteria.

Command	Parameters
ARRAY|ARY SETVIEW	NAME [CRITERIA] [FUNCTION(DELETE|APPEND)]

The following table describes the parameters.

Parameter	Function	Notes
NAME	The name of the array as established during array creation	1- to 31-characters alphanumeric.
CRITERIA	Criteria to which that the row must conform	See comments below this table
FUNCTION	Indicator on how to treat specs	One of the following values: **DELETE** Remove all filter criteria. **APPEND** Append this specification to any already existing specifications. Otherwise, create a new view with the given criteria.

Any number of criteria may be specified, connected by Boolean operators AND and OR. Entries may be of one of the following two formats:

- Boolean operator (AND, OR), except for first criterion when not specifying APPEND.
- Column name, 1-255 characters
- Starting position, default is 1
- Length used for comparison, default width of column+1 - starting position
- Comparison operator, one of the following: LT, LE, EQ, GE, GT,NE,<,<=,=,>=,>,<>,^=
- Column name, 1-255 characters
- Starting position, default is 1
- Length used for comparison, default width of column+1 - starting position

Or

- Boolean operator (AND, OR), except for first criterion
- Column name, 1-255 characters
- Starting position, default is 1

- Length used for comparison, default width of column+1 - starting position
- Comparison operator, one of the following: LT, LE, EQ, GE, GT,NE,<,<=,=,>=,>,<>,^=
- Literal enclosed in quotation marks

The following table describes the TSO/E variables returned from SETVIEW.

TSO/E Variables Returned from SETVIEW			
NAME	**Contents**	**Length/Format**	**Example**
ARYLROWS	Number of rows matching the current view	0-32767 numeric	

The following rules apply:

- When both fields contain numerics (except leading and trailing blanks), a numerical comparison is performed. Both numbers will always be treated as unsigned integers.
- When a comparison with a literal is requested, a pattern comparison is performed (for example, wildcards such as * and + may be used).
- When two columns with different lengths are compared, a comparison with the length of the shorter of the two is done.

Condition codes are listed in the following table.

Value	Description
0	Command was successfully executed.
8	Array not found.
12	Criteria parsing error.
16	Syntax error.

Example

This EXEC connects to the array DASDSTAT and establishes a view of the array which makes only those rows eligible where column STAT has the contents of ACTIVE. It then sorts the resulting array by the contents of column VOL and produces a list of all matching rows.

```
“IMFEXEC ARRAY CONNECT DASDSTAT”
“IMFEXEC ARRAY SETVIEW DASDSTAT CRITERIA(’STAT,,,=,”ACTIVE’’’)”
“IMFEXEC ARRAY SORT DASDSTAT CRITERIA(’VOL,,,A’)”
“IMFEXEC ARRAY INFO DASDSTAT
“IMFEXEC MSG The following volumes are active:”
do i=1 to railways
  “IMFEXEC ARRAY GET DASDSTAT SKIP”
  “IMFEXEC ARRAY MSG “Val
end
```

ARRAY SORT

This command sorts an array according to user specifications. This command is invalid for arrays that were connected using the PAGE keyword.

Command	Parameters
ARRAY|ARY SORT	NAME [CRITERIA(colname,(START),(LENGTH),(ORDER))] [DELETE]

The following table describes the parameters.

Parameter	Function	Notes
NAME	The name of the array as established during array creation	1- to 31-characters alphanumeric.
CRITERIA	Sort criteria to be used	The contents of this parameter are: colname,START,LENGTH,ORDER where **colname** Name of the referenced column in the array **START** Starting position for sort argument (default is 1) **LENGTH** Length of comparison (default is length of column) **ORDER** Ascending or descending Multiple sort arguments may be supplied
DELETE	Removes any array ordering	Default is delete. Removes any sequence binding

After specifying a sort order for an array, the array will be kept sequenced when further insert activity occurs. If high insert activity is expected, it is advisable to remove ordering temporarily, insert all changes, and then respecifiy the sort order. SORT and SETVIEW may be specified in conjunction.

Condition codes are listed in the following table.

Value	Description
0	Command was successfully executed.

4	Criteria parsing error (invalid or missing criteria definition).
8	Array not found.
16	Syntax error.

Example

This EXEC connects to the array DASDSTATS and sorts by the columns STAT and VOL in ascending order.

```
"IMFEXEC CONNECT DASDSTATS"
"ARRAY SORT MYTEST CRITERIA('ROW1,,,A')"
```

Chapter 11. Using the MAINVIEW API

This chapter describes how to use the MAINVIEW API. The MAINVIEW API includes specific commands, functions and facilities that allow AutoOPERATOR users to access data available on the MAINVIEW Databus with AutoOPERATOR EXECs.

Note that the use of this API requires that you are familiar with MAINVIEW AutoOPERATOR IMFEXEC commands and MAINVIEW technology. The API is a cross-platform product which allows you to access data from MAINVIEW technology using AutoOPERATOR automation techniques.

It is also recommended that you have some knowledge about how to use AutoOPERATOR EXECs to access array data. For information about this facility, refer to “Accessing Array Data with AutoOPERATOR EXECs” on page 189.

Overview

The following discussions introduce the MAINVIEW API and describe how to use it.

What Is the MAINVIEW API

The MAINVIEW API allows for a one-way data exchange between MAINVIEW-based products such as MAINVIEW for CICS, MAINVIEW for OS/390, MAINVIEW for IMS (and others) and AutoOPERATOR REXX or CLIST EXECs. Through the API, AutoOPERATOR EXECs can explicitly request the data from a MAINVIEW product through an EXEC. The MAINVIEW API allows AutoOPERATOR to gather data from the MAINVIEW Databus to flow from any of the MAINVIEW products into AutoOPERATOR.

AutoOPERATOR EXECs process the MAINVIEW data based on how a MAINVIEW view looks during a MAINVIEW terminal session. This means you will see the exact same output in an EXEC for a view as you would when it is displayed from a MAINVIEW terminal session.

Data from the MAINVIEW databus is shown in rows and columns (tabular format) so AutoOPERATOR processes MAINVIEW data as an array. An array is a table that consists of one or more columns that are given names. In most instances, an array is processed one row at a time, retrieving the contents of that row into TSO/E variables that are available to an EXEC. There is a one-to-one relationship between the columns of a view and the columns in the resulting array. Refer to “Customize MAINVIEW Views and Connect BBI-SS PAS to a CAS” for more information about the naming conventions for array columns.

Customize MAINVIEW Views and Connect BBI-SS PAS to a CAS

Before AutoOPERATOR EXECs can access MAINVIEW data, you must perform two tasks:

- Customize the MAINVIEW views so that the EXECs can successfully retrieve the column names in variables.
- Connect the BBI-SS PAS to a CAS.

Customizing MAINVIEW Views

There is a relationship between the columns in a MAINVIEW view and the column names of the corresponding array. This relationship is the name of the first header line of a column in a view is the name of the column of the generated array and is also the name of the variable used when retrieving the row of an array (the second header line is ignored).

While this naming convention is intuitive, it poses one major problem: many header lines in MAINVIEW product views do not follow TSO/E variable naming conventions. For example, if a column is titled % TOT CPU, this cannot produce valid variable name because

- It begins with a percent sign (%)
- It contains blanks

The resulting array will show columns with these invalid names but you will still be able to retrieve the rows of a cloumn. The array accepts these invalid names and this allows for simpler debugging when problems arise. You can use the IMFEXEC ARRAY INFO command to display the names of the columns of an array and spot the invalid names.

To resolve these issues, in most cases you must customize a MAINVIEW view to meet a specific need and then update the header lines. The column headers are automatically translated to uppercase. Use the MV CUST facility to create these views which will also allow you to eliminate columns of data that your EXEC is not interested in (such as bar graphs).

Save the customized views and make them available to your BBI-SS PAS by adding a BBVDEF or other DD statement and ensuring that a member in this data set contains the customized view. Refer to the *MAINVIEW Common Customization Guide* and “MAINVIEW VIEW” on page 230 for information about BBVDEF and DD statements.

Understanding Tabular and Detail Views

The MAINVIEW API supports both tabular and detail views.

When you choose to see data from a detail view, only one row containing the requested data is returned. When you choose to see data from a tabular view, the data returned is exactly the same as the width specified by the view customization. Header widths and data widths are independent from each other and you can specify a different width for every column.

Detail views are returned in the same format as they are displayed in the view. A detail view expands the widths of all columns in a row to match the size of the largest row. For example, if you specify three items in one row where one column is 8 characters wide, the second column is 12 characters wide and the third column is 15 characters wide, the data on the screen is aligned so that each item occupies a cell that is 15 characters wide.

The MAINVIEW API follows the same principle. Therefore, if you request data through the API where a row in a detail view is set up as described above, the result creates an array where each variable has a width of 15 characters instead of 3 variables of 8, 12 and 15 characters.

Connecting a BBI-SS PAS to a CAS

The second requirement is to ensure your BBI-SS PAS is connected to a CAS. Specify the ID of the CAS in the BBPARM member BBISSP00 with the CASID= parameter.

When this is finished, you will see the message

```
CT3333I  PAS ssid connected to CAS xxxx
```

during the startup of your AutoOPERATOR BBI-SS PAS where `xxxx` is the name of the CAS you specified with the CASID= parameter.

If you do not see this message, you must determine the cause of the missing connection before you can continue. One thing you can try is to establish a MAINVIEW terminal session, connect with the PAS in question and to invoke some views of the products. If you cannot see data, the products will also be unavailable to the MAINVIEW API.

Once the MAINVIEW views are created and stored and the BBI-SS PAS is communicating with a CAS, you can proceed to writing AutoOPERATOR EXECs.

Using the IMFEXEC MAINVIEW Commands

The following sections provide information about the IMFEXEC MAINVIEW commands and using them with specific parameters.

IMFEXEC MAINVIEW CONNECT

Use the IMFEXEC MAINVIEW CONNECT command to establish a channel.

Channels are the equivalent of a terminal session that interacts with MAINVIEW products. They are a simple abstraction of a terminal session: they do not require you to log on to anything and they do not require any specific definition; you must acquire one.

One advantage of channels is that you can have multiple channels just as if you had set up any number of terminal sessions. Therefore, if your EXEC has to address many different products or views, it is easy to set up more than one channel.

The IMFEXEC MAINVIEW CONNECT command has two parameters: Channel (a required parameter) and MSG (an optional parameter).

The Channel parameter is required to address a specific channel (because you can have more than one) in all IMFEXEC MAINVIEW operations. You can provide the name of a TSO/E variable that will receive a token that uniquely identifies the channel you are addressing. The contents of this variable is used in later IMFEXEC MAINVIEW operations.

Use the MSG parameter to write any error messages to the BBI Journal. During a MAINVIEW terminal sessions you might have experienced a sequence of cascading error messages that explain why a specific operation could not be executed. These exact same statements are turned to an EXEC when an operation failed. By default these messages are added as variables LINE.0 to LINE.xx. In a TSO/E CLIST, the messages are returned as LINE_0 to LINE_xx.

While you can process these error messages programmatically, during development of a new EXEC it may be useful to see them in the BBI Journal without having to explicitly write them out which is what the MSG parameter does. If specified, any error message associated with an operation using this channel is written to the BBI Journal. When the EXEC reaches a production stage, you can remove the MSG parameter to avoid cluttering the Journal.

In summary, a channel is a data transport vehicle requested with the IMFEXEC MAINVIEW CONNECT command and identified by the contents of a variable that you specific.

IMFEXEC MAINVIEW CONTEXT

After a channel is established, point the channel at a particular context with the IMFEXEC MAINVIEW CONTEXT command. The only two parameters absolutely required are the channel you are using and the product you would like to request data from (refer to “MAINVIEW CONTEXT” on page 223 for more information). However you can use this command to point to a specific target or server. By doing this you can use SSI views right out of AutoOPERATOR without having to consolidate the results yourself.

If the target is currently unavailable, you can retry at a later point in time or you can code the WAIT parameter and give the API the opportunity to watch for the availability of your context. If the WAIT times out, you are informed and you can take other actions.

IMFEXEC MAINVIEW VIEW

Once you have gained access to the product, you can access data from the view that you are interested in (just as you would in a regular terminal session) with the IMFEXEC MAINVIEW VIEW command. Refer to “MAINVIEW VIEW” on page 230 for more information.

IMFEXEC MAINVIEW VIEW returns the view name and channel token. The API validates that the view exists and reads its definition.

IMFEXEC MV VIEW will, by default, use views allocated to DDNAME BBVDEF, whether that DDNAME was allocated through JCL or dynamically allocated. This DDNAME can be overridden with the DD(ddname) keyword on the IMFEXEC MV VIEW command.

You might also consider adding view libraries to the currently existing BBIPARM DDNAME concatenation to avoid the need to modify JCL. However, in that case, DD(BBIPARM) must be coded on the IMFEXEC MV VIEW statement.

The data set containing views must have a dataset attribute of LRECL 80.

IMFEXEC MAINVIEW GETDATA

Use IMFEXEC MAINVIEW GETDATA to retrieve the actual data. Refer to“MAINVIEW GETDATA” on page 225.

The two required parameters for this command are the channel name (CHANNEL) and the name of an array (ARRAY). The array is built using the column names you specify and it has as many rows as necessary to accommodate all the data. Make sure the array does not already exist because this command will not overwrite an existing array.

Another parameter that is required the first time you issue this command is REFRESH. If you request data for the first time you must specify this keyword or no data will be returned.

Using the REFRESH parameter is very critical when you need to process subsets of data from a large array. For certain views, a large amount of data may be returned and you will want to process data in subsets. In this case you can use the parameters START and COUNT which allow you to specify (in terms of rows) a subset of data from a view.

As you process these subsets of data, do not use the REFRESH parameter on subsequent operations to ensure the data in the view is constant. When you want fresh data to be retrieved, use the REFRESH parameter again.

If neither the START nor the COUNT parameters are specified, all of the available data is returned.

When the data is available to you in the array, you can process it using IMFEXEC ARRAY statements. Refer to “Accessing Array Data with AutoOPERATOR EXECs” on page 189.

Once you are have retrieved you data and no longer have a use for the channel, it is good practice to release all resources with the IMFEXEC MAINVIEW RELEASE command (“MAINVIEW RELEASE” on page 227). If your EXEC immediately terminates after doing so you can skip the release step because EXEC termination clean up will release the resources for you.

If you do not need the contents of the array anymore, disconnect from it. Remember, arrays can be passed between EXECs and as such termination cleanup will KEEP the array, just in case another EXEC needs it.

The following table lists the IMFEXEC MAINVIEW commands and the page number for additional information.

Command	**Page**	**Function**
CONNECT	221	Request a new channel to be used in subsequent requests.
CONTEXT	223	Connect a channel with a specified context or target.
GETDATA	225	Return collected view data.
RELEASE	227	Release all resources associated with a channel.
TRACE	228	Turn TRACE information on or off.
VIEW	230	Identify the view to be used for accessing data.

General Coding Conventions

The following sections briefly describe the coding conventions for using the IMFEXEC MAINVIEW commands.

Note: Every command described in this chapter is prefixed by the literal MAINVIEW | MV to avoid naming conflicts with existing IMFEXEC constructs. MV is a valid abbreviation.

Using Variable Names

Variable names are limited to 32 characters in length. The first character of the variable must be alphanumeric or one of the following special characters:

- $
- @
- #

Reading Condition Codes

Every command returns a condition code in the variable IMFCC in the TSO pool. Refer to Chapter 4, “Using Variables in REXX EXECs” on page 49 for more information about pools.

Each IMFEXEC command statement description includes a table describing the parameters for the command. The table uses the following format:

Parameter	Function	Notes
1	**2**	**3**

The numbers in this table correspond to the following descriptions:

1 A short parameter identifier. If the parameter has uppercase letters, this identifier must be coded exactly as shown.

If parts of the identifier are shown in **bold**, this parameter can be abbreviated, using the bold letters.

Positional parameters are not associated with a specific identifier. In these cases, this column contains an alias that describes the parameter.

2 The function of the parameter.

3 Notes about the parameter. Typically, these notes describe any length, value, range, or string limitations.

Note: When you invoke a REXX EXEC that has at least one keyword on the PROC statement, you must invoke the EXEC using at least one keyword.

MAINVIEW CONNECT

This command causes a new channel to be used in subsequent channel requests. A channel must be connected before any MAINVIEW data can be requested using the low-level API.

Command	Parameters
MAINVIEW|MV CONNECT	Channel [MSG | NOMSG]

The following table describes the parameters.

Parameter	Function	Notes
Name of channel	Variable name to receive the token that will identify the connected channel.	1- to 32-characters alphanumeric. This variable name will be used by other IMFEXEC MAINVIEW statements
Message option	Controls the writing of exception messages to the journal.	One of the following values: **MSG** Exception messages are written to the journal. **NOMSG** No exception messages are written to the journal (default).

Condition codes are listed in the following table.

Value	Description
0	A new channel was successfully connected and may be referenced by the supplied variable token.
8	A channel could not be acquired. This condition can occur when a BBI-SS PAS or CAS has not been started. Otherwise, use MV TRACE to collect trace information and then contact BMC Customer Support.
16	Syntax error detected during parsing: • Invalid keywords • Missing channel parameter
20	The maximum MAINVIEW session count has been exceeded. The request is failed. The total number of sessions supported PER SS is 150. You might want to retry the request after inserting an IMFEXEC WAIT().

Example

This example shows an EXEC that requests a new channel to be used in subsequent channel requests. The token for this channel is placed into the variable `JOBCHANNEL`.

```
“IMFEXEC MAINVIEW CONNECT JOBCHANNEL”
```

All other examples in this chapter use the variable named JOBCHANNEL to represent the token that identifies the connected channel.

MAINVIEW CONTEXT

This command connects a channel with a specified context or target and optionally waits for it to become available.

Command	Parameters
MAINVIEW|MV CONTEXT	PRODUCT(product name) TARGET(target identifier) [SERVER(server name)] [WAIT(n)] CHANNEL(channelname)

The following table describes the parameters.

Parameter	Function	Notes
PRODUCT	Product to which a connection is to be established.	One of the following products: **MVMVS** MAINVIEW for OS/390 **CMF** CMF Monitor **MVCICS** MAINVIEW for CICS **MVVP** MAINVIEW VistaPoint **MVDB2** MAINVIEW for DB2 **MVIMS** MAINVIEW for IMS **IPSM** MAINVIEW for IMSPlex **MVMQS** MAINVIEW for MQSeries
TARGET	Context or target to which a connection is to be established.	1- to 8-characters alphanumeric.
SERVER	Can be used in target mode to distinguish between different products that contain the same target name.	1- to 8-characters alphanumeric. The default is all servers.

WAIT	Number of minutes to wait until target becomes available.	0 - 99999. The default is 0. If WAIT is specified and a target is not available, a connection is implicitly retried every 10 seconds.
CHANNEL	Token that identifies a previously connected channel.	1- to 32-characters alphanumeric.

Condition codes are listed in the following table.

Value	Description
0	The product or target connection was successfully established. The channel is available to retrieve data from the databus.
8	The connection could not be established.
12	The specified channel could not be located.
16	Syntax error detected or invalid channel token supplied.

Example

This example shows an EXEC that requests an immediate connection to product MVMVS with a target of SJSB. The connection is to use the previously connected channel whose token is contained in the variable `JOBCHANNEL`.

```
“IMFEXEC MAINVIEW CONTEXT PRODUCT(MVMVS) TARGET(SJSB) CHANNEL(“JOBCHANNEL”)”
```

MAINVIEW GETDATA

This command returns some or all of the collected view data.

Command	Parameters
MAINVIEW|MV GETDATA	ARRAY(arrayname) [START(n)] [COUNT(n)] [REFRESH] CHANNEL(channelname)

The following table describes the parameters.

Parameter	Function	Notes
ARRAY	Name of the array in which both the data and the data definition will be returned.	1- to 31-characters alphanumeric. The specified array must not exist. An existing array will not be overwritten. All information about the returned data is implicitly returned in the array.
START	Starting row for the request.	1-99999 numeric. The default is row 1. To request a subset of the data, specify a START value and a COUNT value.
COUNT	Number of rows of data to retrieve.	1-99999 numeric. The default is all rows. To request a subset of the data, specify a START value and a COUNT value.
REFRESH	Specifies that the selector for this data be restored.	The first request for data from a view in a given channel requires REFRESH. In addition: • Always specify REFRESH on the first call. • When using START and COUNT and you are traversing the result set, do not specify REFRESH (because you do not want the result set to change). • When you want a new result set to be obtained (which you always want to unless you are in the situation above), always specify RERESH.

Message option	Controls the writing of exception messages to the journal.	One of the following values: **MSG** Exception messages are written to the journal. **NOMSG** No exception messages are written to the journal (default).
CHANNEL	Token that identifies a previously connected channel.	1- to 32-characters alphanumeric.

Condition codes are listed in the following table.

Value	Description
0	All of the requested data was successfully retrieved.
4	The specified array could not be built because it already exists.
8	The requested data could not be retrieved. Examine the accompanying error messages for details. If NOMSG was specified on CONNECT, display the contents of LINE.xxxx. This return code may also indicate that the START() keyword specified a value that was higher than the number of available records (in which case no records can be returned).
12	The specified channel could not be found.
16	A syntax error was detected or invalid parameters were supplied.
20	An internal error was received.
24	The number of rows returned exceeds the maximum allowed (or 32767). When this condition code is issued, 32767 rows of data will be returned. After processing the returned data, the user can redrive the IMFEXEC MV GETDATA using *ROW(32768)* to obtain any additional rows. When redriving IMFEXEC MV GETDATA to obtain additional rows, do not use REFRESH.

Example

This example shows an EXEC that retrieves data from a previously specified view in the channel called `DATACHANNEL`. For all rows it prints the column with the element name VOL to the AutoOPERATOR journal.

```
"IMFEXEC MAINVIEW GETDATA CHANNEL("DASDCHANNEL") ARRAY(DASDSTAT) START(1) COUNT(20) REFRESH"
"IMFEXEC ARRAY INFO DASDSTAT"
"IMFEXEC MSG The following volumes are active:"
do i=1 to arylrows
  "IMFEXEC ARRAY GET DASDSTAT SKIP"
  "IMFEXEC ARRAY MSG "vol
end
```

MAINVIEW RELEASE

This command releases all resources associated with an API channel.

Command	Parameters
MAINVIEW|MV RELEASE	CHANNEL(channelname)

The following table describes the parameters.

Parameter	Function	Notes
CHANNEL	Token that identifies a previously connected channel.	1- to 32-characters alphanumeric.

Condition codes are listed in the following table.

Value	Description
0	The specified channel was successfully released.
8	An unspecified error occurred while releasing the channel.
12	The specified channel could not be found.
16	Syntax error detected or invalid parameters supplied.

Example

This example shows an EXEC that frees all resources associated with the channel whose token is contained in the variable `JOBCHANNEL`. It then discards any MAINVIEW data returned in the array called `JOBDATA`.

```
"IMFEXEC MAINVIEW RELEASE CHANNEL("JOBCHANNEL")"
"IMFEXEC ARRAY DISC JOBDATA NOSAVE"
```

MAINVIEW TRACE

This command requests trace information to be written to the BBI Journal. Trace information includes the name of the command, the return code, and internal completion and reason codes. Use this command with the generated output whenever contacting BMC Customer Support.

Command	Parameters
MAINVIEW|MV TRACE	ON | OFF

This command is independent of the MSG keyword on the CONNECT statement.

The following table describes the parameters.

Parameter	Function	Notes
ON	Turns MAINVIEW API tracing on.	The ON or OFF parameter must be specified with this command. There is no default value.
OFF	Turns MAINVIEW API tracing off.	The ON or OFF parameter must be specified with this command. There is no default value.

Condition codes are listed in the following table.

Value	Description
0	Tracing was successfully turned ON or OFF
16	Syntax error detected during parsing

Example

This example shows an EXEC that requests that all further MAINVIEW API requests be accompanied by trace information.

```
“IMFEXEC MAINVIEW TRACE ON”
```

MAINVIEW VIEW

This command identifies the view to be used for accessing data.

Command	Parameters
MAINVIEW|MV VIEW	NAME(viewname) [STEM(stemname)] [DD(ddname)] [PARMS(parm1...parm2...parm*n*)] CHANNEL(channelname)

The following table describes the parameters.

Parameter	Function	Notes
NAME	View name that describes the request.	1- to 8-characters alphanumeric. VIEW is an alias for this parameter.
STEM	Stem name of a set of REXX variables containing the view definitions.	1- to 26-characters alphanumeric. This parameter may be used to dynamically specify view contents. A root for a set of stem variables is specified. The variable root.0 contains the total count of stem variables. The actual view is contained in the variables root.1 through root.x. The syntax of the specified view is identical to that of the view normally found in the BBVDEF dataset.
DD	DD name to use to access the view from the BBI-SS PAS.	1- to 8-characters alphanumeric. If a DD name is specified, it must be allocated to the PAS. This can be done either statically or dynamically. If a DD name is specified, the view is read from the DD specified in the calling address space. BBVDEF is the DD that contains the distributed views for accessing AutoOPERATOR data. If a DD name is not specified, the view is accessed from the BBI-SS PAS of the context service point.

PARMS	View parameters as they would be entered on the command line or in a hyperlink.	1- to 80-characters alphanumeric. Parameters in parentheses must be enclosed in quotation marks.
CHANNEL	Token that identifies a previously connected channel.	1- to 32-characters alphanumeric.

Condition codes are listed in the following table.

Value	Description
0	The view was successfully read and parsed. It is available for subsequent GETDATA requests.
4	The specified view could not be read.
8	Bad stem variable specification. The specified variables could not be found.
12	The specified channel could not be found.
16	Syntax error detected or invalid parameters supplied.

Example

This example shows an EXEC that requests that view JOVER be read and parsed. The view will be read from the target BBI-SS PAS. The connection is to use the previously connected channel whose token is contained in the variable `JOBCHANNEL`.

```
"IMFEXEC MAINVIEW VIEW NAME(JOVER) CHANNEL("JOBCHANNEL")"
```

The following example demonstrates how the STEM() parameter may be used to dynamically specify a view:

```
/* REXX */
"ALLOC F(VIEW) DA('BBI26.BAORAE.BBVDEF(PLEX1)') SHR REUSE"
address MVS
"EXECIO * DISKR VIEW (STEM DEFS. FINIS)"
address IMFEXEC
"MV CONNECT MYCHANNEL MSG"
"MV CONTEXT PRODUCT(PLEXMGR) CHANNEL("MYCHANNEL")"
"MV VIEW STEM(DEFS) CHANNEL("MYCHANNEL") VIEW(PLEX1)"
"MV GETDATA CHANNEL("MYCHANNEL") ARRAY(RESULTS) REFRESH"
"MV RELEASE CHANNEL("MYCHANNEL")"
"ARRAY DISC RESULTS NOSAVE
"
```

Sample Program

The sample program in this section illustrates the use of the MAINVIEW API for a complete application.

```
/* rexx */
/*                                                                    */
/**********************************************************************/
/* This EXEC demonstrates the use of the MAINVIEW to AO API.          */
/* It assumes that a customized View -JTEST- exists in a dataset allocated */
/* under the BBVDEF DD statement.                                     */
/* The reason for this is that the names of the columns for the generated */
/* AO array are taken from the HEADER1 columns of the actual BBI-3 View. */
/* AO arrays are processed by taking a row of such an array and introducing */
/* the contents of a row into variable names of the same name. Most BBI-3 */
/* do not lend themselves very well to that purpose since they contain */
/* characters (or even multiple words) that do not translate to individual */
/* variable names (they are invalid variable names).                 */
/* One of the options would have been to make some arbitrary translations. */
/* However, since the user subsequently needs to know the names of such */
/* variables to process them, this would not have been a useful exercise. */
/* It is easy for a user to determine whether invalid variable names exist */
/* by querying the array itself and displaying the column names (which ARE */
/* allowed to be set to names that do not translate to variables). This EXEC */
/* demonstrates this approach amongst other things.                   */
/*                                                                    */
/* All failures of the MV API commands rely on the API's cleanup.     */
/*                                                                    */
/* Note: All MV API commands begin with the prefix -IMFEXEC MV- followed */
/*       by the desired AOI function.                                 */
/**********************************************************************/

/**********************************************************************/
/* The following command turns on MV tracing, a function that causes  */
/* the name of the executed command, the name of the EXEC, return code from */
/* the command as well as API completion and reason code to be automatically */
/* displayed, without having to hand-code it. We may or may not document this*/
/* function to the user.                                              */
/**********************************************************************/

"IMFEXEC MV TRACE ON"

/**********************************************************************/
/* Now we obtain a channel. An AO equivalent (but not identical to) the BBI-3*/
/* token is supposed to be returned in the variable -MYCHANNEL-       */
/**********************************************************************/

"IMFEXEC MV CONNECT MYCHANNEL MSG"
"IMFEXEC MSG 'MVAPICMP: "mvapicmp" MVAPIRSN: "mvapirsn"'"
if rc <> 0 then do
  "IMFEXEC MSG 'MV CONNECT failed'"
  exit
end
```

```
/*****************************************************************************/
/* Establish a connection to the product -MVMVS-, wait a maximum of one      */
/* minute (we automatically retry every minute without the user having to    */
/* specify this number) and use the previously acquired channel (as tokenized*/
/* in the variable -MYCHANNEL-).                                             */
/*****************************************************************************/

"IMFEXEC MV CONTEXT PRODUCT(MVMVS) WAIT(1) CHANNEL("MYCHANNEL")"
if rc <> 0 then do
  "IMFEXEC MSG 'MV CONTEXT failed'"
  exit
end

/*****************************************************************************/
/* Set the proper View -JTEST- using out channel.                            */
/* Please note that unlike the underlying assembler API no information about */
/* the element map is returned. This is deferred until the actual GETDATA    */
/* and then presented in the array structure.                                */
/*****************************************************************************/

"IMFEXEC MV VIEW VIEW(JTEST) CHANNEL("MYCHANNEL")"
if rc <> 0 then do
  "IMFEXEC MSG 'MV VIEW failed'"
  exit
end

/*****************************************************************************/
/* The data is retrieved.                                                    */
/* Any array with the name -RESULTS- is created. The data is refreshed.      */
/*****************************************************************************/

"IMFEXEC MV GETDATA CHANNEL("MYCHANNEL") ARRAY(RESULTS) REFRESH"
if rc <> 0 then do
  "IMFEXEC MSG 'MV GETDATA failed'"
  exit
end

/*****************************************************************************/
/* At this point we want to find out what the names of the columns and their */
/* properties are.                                                           */
/*                                                                           */
/* Note: In the process headers/variable names have been translated to       */
/*       uppercase.                                                          */
/*****************************************************************************/

/*****************************************************************************/
/* We request all pertinent information about the array -RESULTS- and format.*/
/*****************************************************************************/

"IMFEXEC ARRAY INFO RESULTS"
if rc <> 0 then do
  "IMFEXEC MSG 'ARRAY INFO failed'"
  exit
end
```

```
/*****************************************************************************/
/* This is where we format the information. It has been returned by the      */
/* previous command in the variables beginning with the literal -ARY-.       */
/*****************************************************************************/

"IMFEXEC MSG Number of rows:                "arylrows
"IMFEXEC MSG Total storage in use for data: "arystor
"IMFEXEC MSG Number of columns returned:    "arycols
"IMFEXEC MSG Detailed data information follows"
"IMFEXEC MSG ----------------------------------------"
"IMFEXEC MSG Width Name"
"IMFEXEC MSG ----------------------------------------"

/*****************************************************************************/
/* Here we build one line per column that displays its name and width.       */
/* A format of character is assumed.                                         */
/*****************************************************************************/

do i=1 to arycols
  "IMFEXEC MSG "left(arycolw.i,6)||arycoln.i
end

col1=arycoln.1

/*****************************************************************************/
/* At this point we display the complete contents of the array (the returned */
/* BBI-3 data). This is, of course, not advised for very large amounts of    */
/* data.                                                                     */
/*****************************************************************************/

"IMFEXEC MSG Displaying complete ARRAY contents"

/*****************************************************************************/
/* Build a header line that properly names each column and is aligned.       */
/*****************************************************************************/

line=""
do j=1 to arycols
  line=line||left(arycoln.j,arycolw.j+1)
end

"IMFEXEC MSG ----------------------------------------"
"IMFEXEC MSG "line
"IMFEXEC MSG ----------------------------------------"

/*****************************************************************************/
/* Sort by jobname column                                                    */
/*****************************************************************************/

"IMFEXEC ARRAY SORT RESULTS CRITERIA('JOBNAME,,,A')"

/*****************************************************************************/
/* Now retrieve each row, build a line from all column contents and show it. */
/*****************************************************************************/

do i=1 to arylrows
  "IMFEXEC ARRAY FIND RESULTS row("i")"
  "IMFEXEC ARRAY GET "RESULTS
  line=""
```

```
  do j=1 to arycols
    line=line||left(value('arycoln'j),arycolw.j)||' '
  end

  "IMFEXEC MSG "line
end

/*****************************************************************************/
/* Clean the channel up.                                                     */
/*****************************************************************************/

"IMFEXEC MV RELEASE CHANNEL("MYCHANNEL")"
if rc <> 0 then do
  "IMFEXEC MSG 'MV RELEASE failed'"
  exit
end

/*****************************************************************************/
/* We also get rid of the results array. By default the array could be picked*/
/* up at a later point by another EXEC and reprocessed.                      */
/*****************************************************************************/

"IMFEXEC ARRAY DISC RESULTS NOSAVE"

/*****************************************************************************/
/* If you are testing with this EXEC and for some reason, it does not work   */
/* and this last statement is not executed, the next GETDATA will fail,      */
/* indicating the array already exists. The quick remedy is to run           */
/* the EXEC -DELARY- that will delete all disconnected (KEPT) arrays.        */
/*****************************************************************************/
```

Chapter 12. Using the IMFEXEC Statements

IMFEXEC statements provide automation services not available in a TSO command procedure or with a REXX EXEC. The command syntax is the keyword IMFEXEC, followed by the command and any necessary parameters; for example:

```
"IMFEXEC command [parameters]"
```

Valid delimiters for the command are blank characters. IMFEXEC keywords **must** be coded in uppercase.

Command	Page	Function
ALERT	241	Create an exception message in the ALERTS Application
BKPT	259	Used when testing EXECs with the EXEC Testing facility; allows you to set a breakpoint anywhere in the EXEC, including in native REXX code
CHAP	260	Used to change the dispatching priority of the EXEC
CICS	261	Issue a command to a CICS target
CICSTRAN	304	Invoke a transaction in a CICS target
CMD	305	Issue a CICS, IMS, MVS, JES, or BBI command.
CNTL	321	Control listing of EXEC commands in the Journal Log
DOM	323	Delete an outstanding WTO or WTOR
EXIT	324	Terminate the EXEC and set return code
HB	325	Change the number of seconds between heartbeat messages that are sent from a BBI-SS PAS to the ELAN workstation.
IMFC	326	Issue IMF analyzer or monitor command
IMFC SET	331	Issue time-initiated requests from an EXEC
IMSTRAN	333	Initiate an IMS/VS transaction
JES3CMD	336	Issue a JES3 command
JESALLOC	337	Allocate a SYSOUT data set to the given DD name.
JESSUBM	337	Submit a JOB from a DD name or stem variables.
LOGOFF	339	Terminate a previously established OSPI session
LOGON	340	Establish or re-establish an OSPI session between an EXEC and any VTAM application
MSG	342	Write a message in the BBI-SS PAS Journal Log
NOTIFY	343	Initiate a pager request through the Elan workstation
POST	344	Posts a name for an EXEC that waits on that name
RECEIVE	346	Attempt to receive a screen for an OSPI session

Command	Page	Function
RES	347	Issue a SYSPROG service command
SCAN	349	Investigate and retrieve data for an OSPI session
SELECT	352	Invoke an EXEC or user program
SEND	356	Send a message to a TSO or IMS user
SESSINF	358	Write OSPI screen contents and relevant information to the OSPISNAP DD command
SETTGT	359	Set the target system ID
SHARE	360	Exchanges variables with an AOAnywhere EXEC
STDTIME	362	Instruct Elan to get Greenwich date and time and local date and time
SUBMIT	363	Submit a job to MVS
TAILOR	364	Enables you to manipulate the contents of members of partitioned data sets, or REXX stem variables (including a TSO CLIST variation)
TRANSMIT	376	Transmit modified OSPI screen contents to the application
TYPE	378	Enter data into an OSPI session
VCKP	380	Checkpoint PROFILE variables
VDCL	381	Define a variable structure
VDEL	383	Delete variable(s)
VDELL	386	Deletes one or more long variables from one of the AutoOPERATOR variable pools
VDEQ	388	Issue an MVS dequeue
VENQ	389	Issue an MVS enqueue
VGET	391	Retrieve variable(s) from a pool
VGETL	394	Copies one or more long variables from one of the AutoOPERATOR pools into the TSO pool
VLST	395	Retrieve names of defined variable names
VLSTL	397	Retrieves a long variable from the specified pool and places it into the TSO pool
VPUT	399	Store variable in a pool
VPUTL	402	Creates or sets a long variable from a variable in the TSO pool
WAIT	404	Pause for a fixed interval during EXEC processing
WAITLIST	405	Returns information about outstanding WAIT EXECs in variables LINE1 through LINExx
WTO	407	Write a message to the system console
WTOR	410	Write a message to the system console and wait for a reply

General Coding Conventions

The following sections briefly describe the coding conventions for using the IMFEXEC command statements.

REXX Coding

Many of the IMFEXEC keywords contain parentheses. To avoid problems with REXX interpreting IMFEXEC keywords as functions, enclose IMFEXEC statements in double quotation marks:

```
"IMFEXEC ALERT 'CICSPROD has abended' QUEUE(cics)"
```

If you need to use a variable in a REXX IMFEXEC statement, it must not be coded within the double quotes. In the following example, REXX will substitute a value for the variable CQUEUE:

```
"IMFEXEC ALERT 'CICSPROD has abended' QUEUE("cqueue")"
```

In the above example, `"IMFEXEC ALERT 'CICSPROD has abended' QUEUE("` is the first part of the statement, `cqueue` is the value to be substituted, and `")"` is the second part of the statement.

Using Quotation Marks

The IMFEXEC commands conform to TSO CLIST coding conventions; for example, all parameters containing embedded blanks must be enclosed in single quotation marks. To use a single quotation mark in a string of characters, use two single quotation marks.

```
IMFEXEC MSG 'JOB ''I327802'' has abended'            CLIST

"IMFEXEC MSG 'JOB ''I327802'' has abended'"          REXX
```

The resulting message appears in this format:

```
JOB 'I327802' has abended
```

Using Variable Names

Variable names are limited to 32 characters in length. The first character of the variable must be alphanumeric or one of the following special characters:

- $
- @
- #

Reading Condition Codes

Every command returns a condition code in the variable IMFCC in the TSO pool. Refer to “Using Variables in REXX EXECs” on page 49 for more information about pools.

Each IMFEXEC command statement description includes a table describing the parameters for the command. The table uses the format:

Parameter	Function	Notes
1	2	3

The numbers in this table correspond to:

1 A short parameter identifier. If the parameter has uppercase letters, this identifier must be coded exactly as shown.

If parts of the identifier are shown in **bold**, this parameter can be abbreviated, using the bold letters.

Positional parameters are not associated with a specific identifier. In these cases, this column contains an alias that describes the parameter.

2 The function of the parameter.

3 Notes about the parameter. Typically, these notes describe any length, value, range, or string limitations.

ALERT

This command manages exception messages and message queues that can be displayed by any of the STATUS applications and ALERTs applications.

Command	Parameters
ALERT	alert-key 'alert-text' [FUNCTION(ADD|COUNT|CREATEQ|DELETE|DELETEQ|LISTQ|READQ)] [ALARM(NO|YES)] [COLOR(RED|PINK|YELLOW|DKBLUE|LTBLUE|GREEN|WHITE)] [DISPOSE(KEEP|DELETE)] [ESCALATE(UP|DOWN)] [ESCEXEC('execname p1 p2 p3 ... pn')] [EXEC('execname p1 p2 p3 .. pn')] [HELP(panelname)] [INTERVAL(nnnn,nnnn,nnnn,nnnn,nnnn,nnnn)] [PCMD('cmd string')] [POSITION(position)] [PRI(CRITICAL|MAJOR|MINOR|WARNING|INFORMATIONAL|CLEARING)] [PUBLISH(REPLACE|ADD|NO)] [QUEUE(MAIN|queue name)] [RETAIN(YES|NO)] [SYSTEM(YES|NO)] [TARGET(target name)] [TEXT('text string')] [ORIGIN(origin)] [UDATA('user data')] [USER(user name)]

The following table describes the parameters.

Parameter	Function	Notes
alert-key	The key used to uniquely identify an ALERT within a queue	Maximum length is 64 alphanumeric positions. Required for: FUNCTION(ADD) FUNCTION(DELETE) Optional for: FUNCTION(READQ) You must specify a unique key for every ALERT you create. If you create a second ALERT with the same key as an already existing ALERT, the second ALERT will overwrite the first ALERT. The key cannot contain blanks.
'alert-text'	The text of the ALERT message	Maximum message length is 255 alphanumeric positions. Required for: FUNCTION(ADD) If the contents of the text are null but specified (for example, zero length), the ALERT text is replaced by `N/A`. A specification of `/N` within the alert text forces a line break. You must include a blank space before and after using `/N`.
ALARM	Emit audible alarm from the terminal on the ALERT Detail application	Possible values are: YES Sound alarm NO Do not sound alarm NO is the default.

Parameter	Function	Notes
COLOR|COL	The color in which the ALERT is displayed in the ALERT DETAIL and STATUS applications (overrides default color associated with ALERT priority)	This parameter does not have any impact upon the ALERT OVERVIEW application. When an ALERT's priority is increased or decreased (with the ESCALATE parameter), the new ALERT priority's color will always default to the following list of colors: CRITICAL(RED) MAJOR(PINK) MINOR(YELLOW) WARNING(DKBLUE) INFORMATIONAL(LTBLUE) CLEARING(GREEN)
DISPOSE	Allows you to specify whether an ALERT is kept or deleted when it has reached its final escalation priority level	This keyword must be used with the INTERVAL keyword. Possible values are: KEEP Keep the ALERT in its queue DELETE Delete the ALERT from the queue KEEP is the default. The variable AMFEDISP returns the value of this keyword.
ESCALATE	Allows you to create ALERTs that change in priority over a specified interval of time	This keyword must be used with the INTERVAL keyword. Possible values are: UP The ALERT priority is upgraded from less critical to more critical. DOWN The ALERT priority is downgraded from more critical to less critical. UP is the default. The variable AMFEDIR returns the value of this keyword.

Parameter	Function	Notes
ESCEXEC	Allows you to specify an EXEC (with parameters) that is scheduled when the ALERT reaches its final priority level	This keyword must be used with the INTERVAL keyword. The variable AMFEEXEC returns the value of this keyword.
EXEC	The name of the ALERT-initiated follow-up EXEC and its parameters	Maximum length is 256 characters. Refer to “Parameters Passed to the EXEC” on page 31 for more information about parameters passed to ALERT-initiated EXECs.
FUNCTION|FUN	The function to be performed	Use the FUNCTION keyword with: • ADD • COUNT • CREATEQ • DELETE • DELETEQ • LISTQ • READQ For more information about these functions and the return codes they generate, refer to “FUNCTION Names and IMFCC Return Codes” on page 249.
HELP	The name of an extended help panel	Maximum length is 8 characters. This help panel is displayed when you enter the EXPAND primary command in the ALERT DETAIL application while the cursor is positioned on the ALERT. The help panel is a text member without any formatting or control characters. Create a partitioned dataset (LRECL FB 80) to contain your help members. Modify your TSCLIST EXEC to insert this dataset into the PNLLIB concatenation.

Parameter	Function	Notes
INTERVAL	Allows you to specify one to six intervals of time over which the priority of an ALERT will change An ALERT's priority can either increase (become more critical) or decrease (become less critical) in priority over the specified time intervals. The interval can be specified from 0 to 9999 minutes. At least one interval must be specified for an ALERT when ESCALATE is specified. When the final interval expires: • The action specified by the DISPOSE keyword occurs (either the ALERT is deleted or kept) • If an EXEC is specified with the ESCEXEC keyword, the EXEC is scheduled	This keyword **must be used** with the ESCALATE keyword and you must specify at least one interval for an ALERT with ESCALATE specified. The variables AMFEINT1 through AMFEINT6 return the values associated with this keyword. In addition, when you want to have an ALERT change in priority, you must always code one interval more than the number of changes. No priority changes occur in the last interval. For example, if you want an ALERT to change from MAJOR to CRITICAL, you must code two interval periods. Refer to "Examples of ALERT Escalation" on page 254 for examples.
ORIGIN	A new origin to assign to this ALERT	A 1- to 8-character user-defined origin assigned to the ALERT. The first character cannot be a numeric, The user-defined origin overrides the EXEC's IMFSYSID (or the originating job name for the EXEC).

Parameter	Function	Notes
PCMD	A command to be executed if the terminal operator uses the TRANSFER command on the ALERT DETAIL panel	Any command that is valid from the command line is a valid value for this parameter. Maximum length is 256 characters. PCMD is executed as if it were entered on the command line. You should use the SYSTEM parameter (described below) or include the BBI SYSTEM command for ALERTs that contain PCMD to ensure that the target field of the transferred-to application will be correct. If you use the SYSTEM parameter, the SYSTEM command is executed after all other commands specified with PCMD have executed. For example: `PCMD('CICS;EX TRAN;SYSTEM SYSA')` Note that if you have blanks in the PCMD statement, you must use single quote marks.
POSITION|POS	The order of the ALERT in the queue to read	Valid values are in the range from 1 to 32,767. This parameter is used only with the READQ function.
PRIORITY	The priority of the ALERT	A valid value is one of the following: CRITICAL(RED) MAJOR(PINK) MINOR(YELLOW) WARNING(DKBLUE) INFORMATIONAL(LTBLUE) CLEARING(GREEN)

Parameter	Function	Notes
PUBLISH	Specifies whether an ALERT is published and how it is published to connected PATROL Enterprise Manager workstations that have subscribed to receive ALERTs through the General Message Exchange (GME).	Possible values are as follows: REPLACE — An ALERT replace for the ALERT's key/queue is sent to all PATROL Enterprise Manager workstations that have subscribed to receive ALERTs from this AutoOPERATOR. If there is already an ALERT with that key/queue on a PATROL Enterprise Manager workstation, it is deleted before writing the new ALERT with that key/queue. ADD — An ALERT add is sent to all workstations that have subscribed to receive ALERTs from this AutoOPERATOR. If there is already an ALERT with that key/queue on a PATROL Enterprise Manager workstation, it is not deleted before writing the new ALERT with that key/queue. ADD is the default. NO — The ALERT is not written to the connected PATROL Enterprise Manager workstations even if they have subscribed to receive ALERTs.
QUEUE|QUE	The name of the queue to access or into which to place the ALERT	Length can be 1 - 8 characters; embedded blanks are valid.

Parameter	Function	Notes
RETAIN	Allows you to specify that an ALERT will be retained across BBI-SS PAS restarts (both cold and warm restarts) and MVS IPLs Note that using this parameter causes the ALERT to be written to DASD. Therefore, you should use this parameter only after careful consideration. A BBI-SS PAS (warm or cold) start or MVS IPL may eliminate the exceptional situation that caused the ALERT in the first place.	Possible values are: YES Retain this ALERT in disk space so that it can survive a BBI-SS PAS warm or cold start. NO Do not retain this ALERT to survive BBI-SS PAS warm or cold starts. NO is the default. ALERTs that specify RETAIN(YES) cannot also specify the INTERVAL keyword. In other words, ALERTs that are to be retained across BBI-SS PAS restarts or MVS IPLs cannot change priority (either increase or decrease). The variable AMFRTAIN returns the value of this keyword.
SYSTEM	Determines whether or not the ALERT Detail processor switches the current target to the origin of the ALERT when processing a TRANSFER (PCMD)	The default is yes. The target is changed to reflect what was coded in the ORIGIN parameter or the AutoOPERATOR SSID.
TARGET	The target to which the ALERT is sent	The ALERT is sent to the subsystem that manages the specified target and exists only in that subsystem.
TEXT	A pattern text string	This parameter applies to only the READQ and COUNT functions. Only ALERTs matching this text string are considered during these operations.
UDATA	Any desired user data string	Maximum length is 256 bytes. The contents of the UDATA field may be retrieved using the READQ function.
USER	The name of a user ID that the ALERT is addressed to	A 1 - 8 character valid BBI-TS user ID. Contents of the user field can be used to tailorALERT DETAIL displays using the ALERT DETAIL PROFILE panel. Refer to the “ALERT Management Facility” chapter in the *MAINVIEW AutoOPERATOR Basic Automation Guide* for more information.

FUNCTION Keywords

The following table lists in alphabetical order the functions that can be used with the FUNCTION keyword in an IMFEXEC ALERT EXEC statement. The possible return codes from each function are also listed and described in the table.

Table 9. FUNCTION Names and IMFCC Return Codes

FUNCTION	Description	IMFCC Return Code Value	Return Code Description
ADD	Adds an ALERT to a queue and creates a new queue if one does not already exist	0	Add was successful
		8	NODE not found in BBINOD when TARGET used
		12	TARGET not found in BBIJNT
		16	TARGET AutoOPERATOR not available
		20	ALERT queue is full
COUNT	Counts the numbers of ALERTs in a given queue Refer to “TSO Variables Returned from COUNT” on page 252 for more information.	0	COUNT was successful, count value is returned in variable AMFCOUNT
		8	One of the following conditions is true: • Queue does not exist • NODE not found in BBINOD when TARGET used
		12	TARGET not found in BBIJNT
		16	TARGET AutoOPERATOR not available
CREATEQ	Creates a new ALERT queue	0	Queue create was successful
		4	Queue already exists
		8	NODE not found in BBINOD when TARGET used
		12	TARGET not found in BBIJNT
		16	TARGET AutoOPERATOR not available

Table 9. FUNCTION Names and IMFCC Return Codes (Continued)

FUNCTION	Description	IMFCC Return Code Value	Return Code Description
DELETE	Deletes an ALERT by the ALERT key	0	Delete was successful
		4	ALERT does not exist
		8	One of the following conditions is true: • Queue does not exist • NODE not found in BBINOD when TARGET used
		12	TARGET not found in BBIJNT
		16	TARGET AutoOPERATOR not available
DELETEQ	Deletes an ALERT queue	0	Deleteq was successful
		4	Queue does not exist
		8	NODE not found in BBINOD when TARGET used
		12	TARGET not found in BBIJNT
		16	TARGET AutoOPERATOR not available
LISTQ	Lists (in TSO variable IMFNOL) the number of ALERT queues present in the target subsystem Refer to “TSO Variables Returned from LISTQ” on page 252 for more information.	0	LISTQ was successful, ALERT queue data is returned
		8	NODE not found in BBINOD when TARGET used
		12	TARGET not found in BBIJNT
		16	TARGET AutoOPERATOR not available
READQ	Reads an ALERT from the queue and returns the characteristics of the ALERT in TSO variables Refer to “TSO Variables Returned from the READQ Parameter” on page 251 for more information.	0	READQ succesful, ALERT data returned
		4	Either no match found when using KEY and/or TEXT criteria or the search ran past the end of the queue when using the POSITION keyword.
		8	One of the following conditions is true: • Queue does not exist • NODE not found in BBINOD when TARGET used
		12	TARGET not found in BBIJNT
		16	TARGET AutoOPERATOR not available

TSO Variables Returned from the READQ Parameter

The following table lists the TSO variables returned from the READQ parameter.

Name	Contents	Length/Format	Example
AMFALARM	Alarm value of the alert	1 / Y (YES) or N (NO)	Y (for YES)
AMFCOLOR	Color of ALERT	6 / As specified by COLOR parameter	RED
AMFEDIR	Increase or decrease the priority of the ALERT when it is escalated	1 / Character (U or D)	D
AMFEDISP	Keep or delete the ALERT at the final escalation level	1 / Character (K or D)	KEEP
AMFEEXEC	Name of EXEC and EXEC parameters scheduled at final escalation priority	0-256 / Character	ALRTEXEC
AMFEINT1 AMFEINT2 AMFEINT3 AMFEINT4 AMFEINT5 AMFEINT6	Number (in minutes) from 0 to 9999	4 / Numeric (or null)	15
AMFEXEC	EXEC and EXEC parameters associated with the ALERT	0-256 / Character	DBSTART SHIFT2
AMFHELP	Extended Alert member name	8 / Character	HELPXT2
AMFIDATE	Date ALERT was issued	9 / DD-MMM-YY	14-FEB-92
AMFITIME	Time ALERT was issued	8 / hh:mm:ss	12:02:24
AMFKEY	Key of the ALERT	1-64 / Character	DASD01
AMFORGN	Origin of ALERT	1-8 / Character	CICSPROD
AMFPCMD	Primary command specified in ALERT	0-256 / Character	CICS; EX TRAN
AMFPRIOR	Priority of ALERT	13 / As specified in PRIORITY parameter	INFORMATIONAL
AMFPSYS	Value for SYSTEM keyword (could be either YES or NO)	1 / Character (Y or null)	Y
AMFPUB	Value of the PUBLISH keyword when an ALERT is created	2-7/ADD, REPLACE, or NO	ADD
AMFQUEUE	Name of queue for ALERT	8 / Character	MAIN

Name	Contents	Length/Format	Example
AMFRTAIN	Specifies whether or not to retain an ALERT across BBI-SS PAS warm and cold starts	1 / Character (Y or N)	Y
AMFSSID	System from which ALERT was issued	8 / Character	SYSB
AMFTEXT	Text of the ALERT	0-255 / Character	This is a test ALERT
AMFTGT	Target to which ALERT was issued	1-8 / Character	IMS22P
AMFUDATA	User data string	0-256 / Character	Any value specified in UDATA parameter
AMFUSER	Name of the user ID that the ALERT is addressed to	8 / Character	JDB1

TSO Variables Returned from COUNT

The following table lists the TSO variables returned from the COUNT parameter.

Name	Contents
AMFCOUNT	Number of ALERTs in designated queue

TSO Variables Returned from LISTQ

The following table lists the TSO variables returned from the LISTQ parameter.

Name	Contents
IMFNOL	Number of queues present in the target susbsystem. In variables LINE1 through LINExxx, it returns the names of all the queues. Limit is 500 queue names.

Examples

This section describes examples using the IMFEXEC ALERT command statement. A brief discussion follows the example.

Example 1: Creating a Multi-line ALERT

```
/* REXX */
"IMFEXEC ALERT NETW2",
"'COMMUNICATION LINES DOWN:  /N   - DALLAS /N +  - CHICAGO'  FUNCTION",
"(ADD) QUEUE(NETWORK)",
"PRIORITY(CRITICAL) COLOR(PINK)"
```

ALERTs are created as single-line messages unless you use the characters `/N` in the alert-text parameter. The characters `/N` indicate the beginning of a new line of alert-text.

You must use a blank space before and after `/N`. In the example above, the alert-text parameters includes the use of `/N` in two places. The EXEC command in this example produces the following multi-line ALERT:

```
___ 11:43     CHICAGO  COMMUNICATION LINES DOWN:
                                - DALLAS
                                - CHICAGO
```

Example 2: Associating a Help Panel with an ALERT

```
/* REXX */
"IMFEXEC ALERT NETW1",
 "'ALM0100 - 8100 COMMUNICATION LINE DOWN: /N        - CHI998A21'",
 "FUNCTION(ADD) QUEUE(NETWORK) PRIORITY(WARNING) HELP(H8100)",
 "COLOR(RED)"
```

Use the HELP keyword of the IMFEXEC ALERT command statement to indicate there is a help panel associated with an ALERT.

Prior to using the HELP keyword in the IMFEXEC ALERT command, you must create and add the help panel to BBPLIB. The HELP keyword specifies the name of the BBPLIB member name. The example shows an IMFEXEC ALERT command statement that specifies a help panel named H8100. The example is a REXX statement and therefore uses double quotation marks. The ALERT created by the EXEC appears on the ALERT DETAIL panel in the following format:

```
TIME  IND ORIGIN _____________________________________
11:44 h   CHICAGO  ALM0100 8100 COMMUNICATION LINE DOWN:
                         -CHI998A21
```

The ALERT displays with an `h` in the `IND` column. This indicates that there is a help panel associated with the ALERT.

To access the help panel, place the cursor anywhere on the ALERT text and press the PF key assigned to EXPAND. You can also type EXPAND on the command line and then place the cursor anywhere on the ALERT text and press ENTER.

Example 3: Managing ALERT Queues

```
/* REXX */
"IMFEXEC VGET THRSHOLD"
"IMFEXEC ALERT  FUNCTION(COUNT) QUEUE(NETWORK)"
N=AMFCOUNT
DO WHILE N > 0
  "IMFEXEC ALERT FUNCTION(READQ) QUEUE(NETWORK) POSITION("N")"
  IF IMFCC = 0 THEN DO
    IF AMFUDATA > THRSHOLD THEN DO
      "IMFEXEC ALERT "AMFKEY" FUNCTION(DELETE) QUEUE(NETWORK)"
      "IMFEXEC ALERT "AMFKEY" FUNCTION(ADD) '"AMFTEXT"' QUEUE(SUPERVSE)"
    END
  END
  N = N - 1
END
```

You can periodically check the queues for ALERTs that have not been responded to and change their priority so that they are noticed.

In the above EXEC, the READQ function is used to set AMFCOUNT equal to the number of ALERTs in the Network queue. The EXEC then reads each ALERT from the NETWORK queue using POSITION and tests the user data presented in the AMFUDATA variable.

If the criteria is met, the ALERT is deleted from the Network queue using the AMFKEY variable (the key of the ALERT). Then the ALERT is added to the supervisor's queue using the same key and using the original text in the AMFTEXT variable.

Note: This example assumes that the ALERTs were originally created with some meaningful user data (such as the date and time).

Examples of ALERT Escalation

The following examples show how to create ALERTs with the ESCALATE parameter so that an ALERT can increase or decrease in priority over a specified interval(s) of time.

Example 1: Escalating an ALERT from lowest to highest priority: The ALERT in this example will be upgraded from Informational to Critical priority over five time intervals. The following list describes the properties of the ALERT:

- The original priority of the ALERT is Informational (`PRIORITY(info)`).
- The ALERT's priority will be upgraded (`Escalate(up)`).
- The priority will be upgraded gradually over the intervals of 10 minutes, 20 minutes, 30 minutes, 30 minutes, and 40 minutes (`Interval(10,20,30,30,40)`).
- When the ALERT reaches the final priority level, the ALERT should be deleted (`Dispose(delete)`).

```
/* REXX */

"IMFEXEC ALERT key1 'test alert' Priority(info) Escalate(up)",
         "Interval(10,20,30,30,40)  Dispose(delete)"
```

When the EXEC that schedules this ALERT is scheduled, the ALERT's original priority is Informational. After 10 minutes ( ), the priority automatically is upgraded from Informational to Warning. The ALERT stays at the Warning priority for 20 minutes ( ) and is upgraded to Minor. The ALERT stays at Minor priority for 30 minutes ( ) before being upgraded to Major. It stays at Major priority for 30 minutes ( ) before being upgraded to Critical. After remaining at Critical for 40 minutes ( ), the ALERT is deleted.

Example 2: Downgrading ALERT priority over two intervals: The ALERT in this example will be downgraded over two time intervals. The following list describes the properties of the ALERT:

- The original priority of the ALERT is Minor (`PRIORITY(minor)`).
- The ALERT's priority will be downgraded (`Escalate(down)`).
- The priority will be downgraded over the intervals of 10 minutes and 20 minutes (`Interval(10,20)`).
- When the ALERT reaches the final priority level, the ALERT should be deleted (`Dispose(delete)`).

```
/* REXX */

"IMFEXEC ALERT key2 'test alert' Priority(minor) Escalate(down)",
         "Interval(10,20) Dispose(delete)"
```

When the EXEC that schedules this ALERT is scheduled, the ALERT's original priority is Minor. After 10 minutes ( ), the priority automatically is downgraded from Minor to Warning. The ALERT remains at the Warning priority for 20 minutes ( ) and is deleted at the end of the interval.

The intervals in this example also can be validly coded as follows:

```
Interval(10,20,)
```

or

```
Interval(10,20,,)
```

or

```
Interval(10,20,,,,)
```

Example 3: Upgrading an ALERT and scheduling an escalation EXEC: The ALERT in this example will be upgraded over two time intervals and, at the end of the second interval, an escalation EXEC will be scheduled. The following list describes the properties of the ALERT:

- The original priority of the ALERT is Minor (`PRIORITY(minor)`).
- The ALERT's priority will be upgraded (`Escalate(up)`).
- The priority will be upgraded over the intervals of 10 minutes and 20 minutes (`Interval(10,20)`).
- When the ALERT reaches the final priority level, the ALERT should be kept until it is manually deleted (`Dispose(keep)`).

- When the ALERT completes its final interval, an EXEC named e100 with three parameters is scheduled (`Escexec('e100 p1 p2 p3')`).

```
/* REXX */

"IMFEXEC ALERT key2 'test alert' Priority(minor) Escalate(up)",
         "Interval(10,20) Dispose(keep) Escexec('e100 p1 p2 p3')"
```

When the EXEC that schedules this ALERT is scheduled, the ALERT's original priority is Minor. After 10 minutes ( ), the priority automatically is upgraded from Minor to Major. The ALERT remains at the Major priority for 20 minutes ( ) and the EXEC e100 with its three parameters is scheduled at the end of the interval. The ALERT remains at the Major priority until it is manually deleted.

Example 4: Skipping ALERT priorities during ALERT escalation: The ALERT in this example will be upgraded from Informational to Major while skipping the intermediate ALERT priorities. The following list describes the properties of the ALERT: •

- The original priority of the ALERT is Informational(`PRIORITY(info)`).
- The ALERT's priority will be upgraded (`Escalate(up)`).
- The priority will be upgraded over the two intervals of 10 and 20 minutes.

  However, to skip ALERT priorities, you must specify an interval of zero minutes for each of the intervals you want to skip.

In this example, the ALERT will skip two priorities and change from Informational priority directly to Major after a 10-minute interval (`Interval(10,0,0,20)`).

- When the ALERT reaches the final priority level, the ALERT should be kept until it is manually deleted (`Dispose(keep)`).
- When the ALERT completes its final interval of 20 minutes, an EXEC named e100 with three parameters is scheduled (`Escexec('e100 p1 p2 p3')`).

```
/* REXX */

"IMFEXEC ALERT key2 'test alert' Priority(info) Escalate(up)",
         "Interval(10,0,0,20) Dispose(keep) Escexec('e100 p1 p2 p3')"
```

When the EXEC that schedules this ALERT is scheduled, the ALERT's original priority is Informational. After 10 minutes ( ), the ALERT's priority automatically is upgraded from Informational to Major. To skip the intermediate priorities, you must code zero minutes for both Warning and Minor priorities (and).

The ALERT remains at the Major priority for 20 minutes ( ) and the EXEC e100 with its three parameters is scheduled at the end of the interval. The ALERT remains at the Major priority until it is manually deleted.

The intervals in this example also can be validly coded as:

```
Interval(10,0,0,20,)
```

or

```
Interval(10,0,0,20,,)
```

Example 5: Showing the elapsed time for an escalated ALERT The ALERT in this example will be upgraded from Minor to Major in one 10-minute interval. The following list describes the properties of the ALERT:

- The original priority of the ALERT is Minor (`PRIORITY(minor)`).
- The ALERT's priority will be upgraded (`Escalate(up)`).
- The priority will be upgraded over one interval of 10 minutes (`Interval(10)`).
- When the ALERT reaches the final priority level, the ALERT should be deleted (`Dispose(delete)`).
- When the ALERT completes its final interval, an EXEC named e100 with three parameters is scheduled (`Escexec('e100 p1 p2 p3')`).

```
/* REXX */

"IMFEXEC ALERT key2 'test alert' Priority(minor) Escalate(up)"
         "Interval(10,20) Dispose(delete) Escexec('e100 p1 p2 p3')"
```

The following example shows the life of the ALERT over time:

```
1:00pm                    1:10pm                          1:30pm
A Minor ALERT       -->   The ALERT is upgraded   -->     The ALERT is deleted
is created                to Major Priority               and the EXEC e100
                                                          is scheduled
The ALERT stays at this   The ALERT stays at this
priority for 10 minutes    priority for 20 minutes
```

Examples of Invalid Coding with the Interval Parameter

Some examples of invalid coding are:

Example 1: The interval keyword must contain at least one value.

```
"IMFEXEC ALERT key4 'test alert' Priority (info) Escalate(up) Interval(,)"
```

Example 2: You can only specify as many intervals as there are between an originating priority and the end priority.

```
"IMFEXEC ALERT key4 'test alert' Priority(major) Escalate(up)
Interval(10,10,10)"
```

In this example, there is only one priority that a major ALERT can be upgraded to (Critical) and yet three intervals are specified.

Example 3: The interval keyword cannot have null values for intervals.

```
"IMFEXEC ALERT key4 'test alert' Priority(major) Escalate(up)
Interval(,10,10)"
```

or

```
"IMFEXEC ALERT key4 'test alert' Priority(info) Escalate(up)
Interval(,10,,20)"
```

Example 4: The intervals cannot have negative values.

```
IMFEXEC ALERT key4 'test alert’ Priority(info) Escalate(up) Interval(,10,-20)
```

Examples of the PUBLISH Keyword

The following examples demonstrate the usage of the IMFEXEC ALERT PUBLISH keyword.

Example 1: This example creates an ALERT and publishes it to all connected PATROL Enterprise Manager workstations, deleting any ALERTs already present with the same queue name and key.

```
IMFEXEC ALERT TESTKEY ‘THIS IS A TEST’ FUNCTION(ADD) PUBLISH(REPLACE) QUEUE(TEST AREA)
```

Example 2: This example creates an ALERT but does not publish it to any connected PATROL Enterprise Manager workstation.

```
IMFEXEC ALERT TESTKEY ‘DO NOT PUBLISH ME’ FUNCTION(ADD) PUBLISH(NO) QUEUE(MAIN)
```

BKPT

Use this command anywhere in an EXEC when you want to set a breakpoint. The breakpoint marks where the EXEC will stop while it is being executed by the online EXEC Testing facility. If you execute the EXEC outside of the online EXEC Testing facility, this command has no effect.

You can use this statement in native REXX code.

Command	Parameters
BKPT	

This command has no parameters. Use of this command has no effect on the value of variable IMFCC.

CHAP

This command uses a specified numeric parameter to change the dispatching priority of the EXEC either up or down.

Command	Parameters
CHAP	(n)

The following table describes the parameters.

Parameter	Function	Notes
n	A numerical value that changes the dispatching priority (either up or down) of the EXEC. The value you specify is added to the current dispatching priority.	The numerical value can range from -255 to 255. After the EXEC terminates, the new dispatching value is returned in the variable IMFPRIO. The value of IMFPRIO can be from 0 to 255.

Condition codes are listed in the following table.

Value	Description
16	Syntax error

Example

This example shows IMFEXEC CHAP where the specified value (-10) will be added to the current dispatching priority.

```
IMFEXEC CHAP(-10)
```

Specifying a value of zero (0) returns the current priority.

CICS

The IMFEXEC CICS command statements use additional commands to manage and control CICS resources.

An IMFEXEC CICS command statement consists of the keyword IMFEXEC, the command prefix CICS, and an AutoOPERATOR or MAINVIEW for CICS command with additional parameters. You can specify resources with generic (*) and positional (+) wild card characters, except when noted. Note that when you use a generic in some resource names, a maximum of only 200 discrete commands are executed. See each command for which ones are affected by this limitation.

CICS requests only indicate success or failure of the scheduling of the service. The AutoOPERATOR for CICS component within the CICS address space issues additional messages to indicate its success or failure. You should code Rule-initiated EXECs (triggered by journal messages in the format FTxxx) to process the responses from CICS to ensure successful completion of CICS-dependent commands.

The IMFEXEC CICS commands are supported only on a CICS system that is defined to the local AutoOPERATOR BBI-SS PAS. If a CICS target is used (refer to IMFEXEC SETTGT command on page 359) that is defined to a remote AutoOPERATOR BBI-SS PAS, you will receive a FT421S message and the service will fail.

To avoid this, use the IMFEXEC SELECT command to schedule an EXEC on a remote AutoOPERATOR BBI-SS PAS and that EXEC can issue the CICS command.

Condition Codes

The following table describes condition codes returned after issuing an IMFEXEC CICS command statement.

Value	Meaning
0	Normal completion
4	Warning condition; not necessarily an error
8	Exceptional condition
12	Error condition; did not complete operation. Possible reasons are: • For independent actions, the region was not available • For dependent functions, the region was not connected to the BBI-SS PAS
16	Error condition
20	Severe error condition

The IMFCC variable can be tested by commands in the EXEC that follow the IMFEXEC command. However, the IMFCC condition code is different for services that are dependent or services that are independent of CICS.

For CICS-dependent services (where CICS performs the task), the request is routed to the respective CICS system for processing. If the request is successfully scheduled, IMFCC is set to 0; if the request fails, IMFCC is set to 8. Either message FT037I or FT038W is written to the Journal log at this time. The final status of the service is written to the Journal log by messages FT401 through FT414. These messages are accompanied with explanatory text. CEMT is an exception; CEMT returns the actual CICS response to the log instead of issuing FTxxxx messages.

For CICS-independent services (services that do not require CICS to perform the task), a Service Request Block (SRB) is scheduled to the target CICS system to perform the processing. If the request is successfully scheduled, IMFCC is set to 0; if the request fails, IMFCC is set to 8. Either message FT037I or FT038W is written to the Journal log at this time. The final status of the service is written to the Journal log by messages FT401 or FT414. These messages are accompanied with explanatory text.

See Table 10 for details on which services are CICS dependent and which are not.

CICS Command Parameters

Services marked as dependent require that BBI-SS PAS to CICS communication is active. Refer to the *MAINVIEW AutoOPERATOR Customization Guide* for details.

Table 10. List of IMFEXEC CICS Command Statements

Command	Function	Dependent	Page
ACQUIRE	Acquire a VTAM-supported terminal	Yes	263
ALLOC	Allocate a data set	Yes	266
ALTER	Alter a CICS task-related value	Yes	267
ALTERVS	Alter virtual storage	No	273
CEMT	Issue a CICS extended master terminal command	Yes	274
CHAP	Change a task's priority	Yes	275
CICSKEY	Change CICSKEY settings for CIS transactions	No	276
CLOSE	Close a file	Yes	277
CONN	Alters the status of IRC/ISC connections	Yes	278
DISABLE	Disable a resource	Mixed[1]	279
DROP	Decrease the use count of a program	Yes	281
DUMPDB	Prepare a database for dumping	Yes	282
ENABLE	Enable a resource	Mixed[1]	283
FREE	Deallocate a file	Yes	285
INSERVE	Place a resource in service	Yes	286
ISOLATE	Change ISOLATE settings for CIS transactions	No	287
KILL TASK	Terminate a CICS task by task number	Mixed[1]	288
KILL TERM	Terminate a CICS task by terminal	Yes	290
LOAD	Load a program	Yes	291
NEWCOPY	Load a new version of program	Yes	292
OPEN	Open a file	Yes	293
OUTSERVE	Take a resource out of service	Yes	294
PURGE	Purge a resource	Yes	295
QUERY	Invoke a MAINVIEW for CICS service	No	297
RECOVERDB	Prepare a database for recovery	Yes	299
RELEASE TERMINAL	Release a VTAM terminal	Yes	300

Table 10. List of IMFEXEC CICS Command Statements (Continued)

Command	Function	Dependent	Page
SPURGE	Change the SPURGE value for a CICS transaction	No	301
STARTDB	Start a database	Yes	302
STOPDB	Stop a database	Yes	303

[1] Some common options are dependent. Refer to the description of each command for more information.

CICS ACQUIRE

This command issues a VTAM request to acquire a terminal.

Command	Parameters
CICS ACQUIRE	TERMINAL Terminal identifier

The following table describes the parameters.

Parameter	Function	Notes
Terminal	The terminal to be acquired	1- to 4-alphanumeric. If you use generics for the terminal ID name, a maximum of only 200 discrete commands are executed.

Note: BBI-SS PAS to CICS communication must be active.

Example

The commands in this example acquire terminals following CICS startup without needing to specify CONNECT=AUTO in the Terminal Control Table (TCT). WAIT was specified to minimize the impact on CICS processing.

```
/* REXX */
"IMFEXEC CICS ACQUIRE TERMINAL AB00"
"IMFEXEC CICS ACQUIRE TERMINAL AB01"
"IMFEXEC CICS ACQUIRE TERMINAL AC00"
"IMFEXEC CICS ACQUIRE TERMINAL AC01"
     .
     .
     .
"IMFEXEC WAIT 5
"IMFEXEC CICS ACQUIRE TERMINAL BA11"
"IMFEXEC CICS ACQUIRE TERMINAL BA12"
```

CICS ALLOC

This command allocates a file or data set to either the CICS region or to the BBI-SS PAS. The allocation is done shared (DISP=SHR).

Command	Parameters
CICS ALLOC	Filename [TO] Dsname [LOCAL]

The following table describes the parameters.

Parameter	Function	Notes
Filename	The DD Name of the file to allocate	Length can be 1- to 8 alphanumeric. An FCT entry is not needed.
TO	Readability token	
DSName	Name of data set to allocate	1-44 characters alphanumeric.
LOCAL	Forces allocation to the BBI-SS PAS instead of the CICS region	

Example

This example command allocates a data set that has previously been freed for batch processing.

```
/* REXX */
"IMFEXEC CICS ALLOC MASTER TO USER.VSAM.MASTER"
```

CICS ALTER

This command allows changing of the values of the CICS class maximum settings and statistics for each class.

Note: For CICS/ESA 4 and above, BBI-SS PAS to CICS communication must be active

Command	Parameters
CICS ALTER	MAXTASK | ICV | ICVR | CLASSn | TCLASS | SYSTEM | DUMPDS | TCPIPSERVICE | JVMPOOL

The following table describes the parameters.

Parameter	Function	Notes

MAXTASK	Specifies the maximum number of tasks, active and suspended, allowed in the CICS address space concurrently.
	
	Example:
	
	`"IMFEXEC CICS ALTER MAXTASK 35"`
	`/* Allow only 35 tasks to run */`
		Values can be 1 - 999.

ICV	Specifies the region exit interval value in milliseconds.
	
	Example:
	
	`"IMFEXEC CICS ALTER ICV 1000"`
	`/*Come back from OS after 1 second*/`
		Values can be 100 - 3600000.

ICVR	Specifies the runaway interval in milliseconds.
	
	Example:
	
	`"IMFEXEC CICS ALTER ICVR 5000"`
	`/* If it runs longer than 5 seconds, it is looping */`
		Values can be 500 - 2700000.

CLASS1 - CLASS10	Specifies the largest number of tasks in this class that can be active concurrently.
	
	Example:
	
	`"IMFEXEC CICS ALTER CLASS1 10"`
	`/* Allow only 10 tasks in this class */`
		Values can be 1 - 999.

Parameter	Function	Notes

TCLASS class
 	
MAXACTIVE value | PURGETHRESH value
	Reset the maximum number of tasks or the purge threshold for a transaction class.
	
	Example 1:
	
	`"IMFEXEC CICS ALTER TCLASS DFHTCL05 MAXACTIVE 200" /* Only allow 200 tasks to run */`
	
	Example 2:
	
	`"IMFEXEC CICS ALTER TCLASS DFHTCL05 PURGETHRESH 1000" /* Only allow 1000 tasks to queue up */`
		Possible attributes and values are
		
		Class
		Any valid 1 - 8 character transaction class name. The word class is not a keyword. It indicates where the positional parameter class name is specified.
		
		MAXACTIVE Value
		Valid values can be 0 - 999. Cannot be specified with PURGETHRESH. After the value for class, specify MAXACTIVE followed by a value.
		
		PURGETHRESH Value
		Valid values can be 0 - 1000000. Cannot be specified with MAXACTIVE. After the value for class, specify PURGETHRESH followed by a value. Only one attribute can be changed per execution of the statement. You cannot change both the MAXACTIVE and the PURGETHRESH attribute with the same statement.
		
		ALTER TCLASS is only available for CTS 1.3 and later.

Parameter	Function	Notes

SYSTEM attribute value
 	
Possible attributes are
 	
AKP
DSALIMIT
DTRPROGRAM
DUMPING
EDSALIMIT
FORCEQR
PROGAUTOINST
PROGAUTOCTLG
PROGAUTOEXIT
PRTYAGING
RUNAWAY
SCANDELAY
TIME
	Issues commands to change certain CICS system attributes. Attribute specifies the system attribute and value is the desired value.
	
	Example 1:
	
	`"IMFEXEC CICS ALTER SYSTEM DUMPING NO"`
	`/* This command disallows dumps */`
	
	Example 2:
	
	`"IMFEXEC CICS ALTER SYSTEM DSALIMIT 8388608"`
	`/* Set DSA limit to 8 megabytes*/`
	
	Example 3:
	
	`"IMFEXEC CICS ALTER SYSTEM EDSALIMIT 500M"`
	`/* Set EDSA limit to 500 megabytes */`
	
	Example 4:
	
	`"IMFEXEC CICS ALTER SYSTEM EDSALIMIT 1G"`
	`/* Set EDSA limit to 1 gigabytes */`
		Possible attributes and values are
		
		AKP
		Valid range is 200 - 65535. Specifies the activity keypoint trigger value, which is the number of write requests to the CICS system log stream output buffer between the keypoints. The value 0 is also valid and specifying it turns off keypoints.
		
		DSALIMIT
		Valid values are 2MB - 16MB. Specifies the maximum amount of dynamic storage area CICS can allocate below the 16 megabyte line. Values can be specified in bytes, kbytes or mbytes by appending K or M to the end of the value, or by leaving a blank for bytes.
		
		DTRPROGRAM
		Specifies the Dynamic Routing program name.
		
		DUMPING
		Valid values are YES and NO. Indicates whether CICS system dumps can be taken.
		
		EDSALIMIT
		Valid values are 10M - 2G. Specifies the maximum amount of dynamic storage area CICS can allocate above the 16 megabyte line. Values can be specified in bytes, kbytes, mbytes or gbytes by appending K, M or G to the end of the value, or by leaving blank for bytes.
		
		FORCEQR
		Valid values are FORCE and NOFORCE. Specifies whether you want CICS to force all user application programs specified as CONCURRENCY(THREAD-SAFE) to run under the CICS QR TCB, as if they were specified as CONCURRENCY(QUASIRENT) programs. SYSTEM FORCEQR is available only for CTS 1.3 and later.

Parameter	Function	Notes
SYSTEM attribute value (Continued)		See the *CICS System Programming Reference Guide* for more information. PROGAUTOINST Valid values are ACTIVE and INACTIVE. Specifies whether autoinstall for programs is to be active or inactive. PROGAUTOCTLG Valid values are NONE, ALL or MODIFY. Specifies which autoinstalled program definitions are to be cataloged and when. Definitions are to be cataloged only when modified. PROGAUTOEXIT Specifies the name of the user-provided program to be called by the CICS program autoinstall code to provide a model definition. PRTYAGING Valid values are between 0 and 65535 (in milliseconds). Specifies the rate at which CICS is to increase the priority of a task waiting for dispatch. RUNAWAY Valid values are between 500 and 2700000 (in milleseconds). Specifies the default for runaway task time. SCANDELAY Valid values are 0 to 5000 (in milliseconds). Specifies the maximum number of milliseconds between a user task making a terminal I/O request and CICS dispatching the terminal control task to process it. TIME Valid values are in the range 100 - 3600000. Specifies the maximum interval in milliseconds for which CICS gives control to the operating system if no tasks are ready for dispatch. Only one attribute can be changed per execution of the statement. You will need to code multiple statements in order to change multiple attributes.

Parameter	Function	Notes

DUMPDS attribute value
 	
Possible attributes are
 	
DATASET
INITIALDDS
OPENSTATUS
SWITCHSTATUS
	Changes the attributes of the CICS dump data set.
	
	Example 1:
	
	`"IMFEXEC CICS ALTER DUMPDS OPENSTATUS OPEN"`
	`/* This command opens the active dump dataset */`
	
	Example 2:
	
	`"IMFEXEC CICS ALTER DUMPDS INITIALDDS AUTO"`
	`/* Next warm start use whichever dataset not used last */`
		Possible attributes and values are
		
		DATASET
		Valid values are A and B. Specifies current dump data set.
		
		INITIALDDS
		Specifies which dump data set is to be active first on subsequent warm or emergency restarts. Valid values are A, B and AUTO. AUTO indicates to use the data set that was not active when CICS last terminated (normally or abnormally).
		
		OPENSTATUS
		Valid values are OPEN and CLOSE. Specifies actions to be taken on the transaction dump data sets.
		
		SWITCHSTATUS
		Valid values are NO and NEXT. Specifies whether CICS is to switch active data sets automatically the next time the current dump data set fills up.

Parameter	Function	Notes

TCPIPSERVICE service attribute value
 	
Possible attributes are
 	
BACKLOG
DNSSTATUS
STATUS
URM
	Modify the status of a service using CICS internal TCP/IP support.
	
	Example 1:
	
	`"IMFEXEC CICS ALTER TCPIPSERVICE PRINTER STATUS CLOSE"`
	`/* Close printer service */`
		Possible attributes and values are
		
		BACKLOG
		Changes the maximum number of requests that can be queued in TCP/IP waiting to be processed by the service. Specify service name followed by BACKLOG followed by value.
		
		DNSSTATUS
		Valid values are REGISTERED and DEREGISTERED. Changes the Domain Name System (DNS)/Workload Manager (WLM) registration status of this service. Specify service name followed by DNSSTATUS followed by value.
		
		STATUS
		Valid values are OPEN, CLOSE and IMMCLOSE. Changes the status of the service. Specify service name followed by STATUS followed by value.
		
		URM
		Specifies the 8-character name of the program to be used as the Service User-replaceable module. Specify service name followed by URM followed by value.
		
		ALTER TCPIPSERVICE is only available for CTS 1.3 and later.

JVMPOOL attribute value
 	
Possible attributes are
 	
STATUS
TERMINATE
	Enable or disable the JVM pool, or terminate the pool altogether.
	
	Example 1:
	
	`"IMFEXEC CICS ALTER JVMPOOL STATUS DISABLED"`
	`/* No new requests are allowed */`
	
	Example 2:
	
	`"IMFEXEC CICS ALTER JVMPOOL TERMINATE PURGE"`
	`/*Purge all the tasks */`
		Possible attributes and values are
		
		STATUS
		Valid values are ENABLED and DISABLED. Specifies whether new Java requests can be accepted and serviced by the JVM pool.
		
		TERMINATE
		Valid values are PHASEOUT, PURGE and FORCEPUR. Specifies that the JVM pool is to be terminated.
		
		ALTER JVMPOOL is only available for CTS 2.1 and later.

Example

Examples are located in the Parameters table with the description of each keyword.

CICS ALTERVS

This command allows changing of the contents of memory located at the specified virtual address.

Command	Parameters
CICS ALTERVS	Address [FROM] Value1 [TO] Value2

The following table describes the parameters.

Parameter	Function	Notes
Address	A virtual storage address	8 hexadecimal digits (4 bytes).
FROM	Readability token	Used primarily for documentation purposes; however, it must be different from the TO value to cause the storage to be altered.
Value1	Current memory contents at the designated virtual storage address	8 hexadecimal digits.
TO	Readability token	
Value2	Replaces the current contents of memory at the specified virtual storage address with a new hexadecimal value	8 hexadecimal digits (4 bytes).

Example

This example command zeroes out a field known to be in a specific location in a control block or program.

```
/* REXX */
"IMFEXEC CICS ALTERVS 00031F14 FROM 01080000 TO 00000000"
```

CICS CEMT

This command issues a CICS CEMT request.

Command	Parameters
CICS CEMT | CEMTQ	Mttran The maximum length of parameters (including blanks) and subcommands (such as SET and INQUIRE) is 72 characters. By default, the output from the CEMT command is written to the BBI journal. If you many CEMT commands consecutively or over time, it can produce a great deal of unwanted data in the BBI journal. To avoid this overload of information, you can use the CEMTQ command instead of CEMT. All parameters are specified exactly as with CEMT. The difference is that the output will not be written to the BBI journal.

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
Mttran	Is a CICS master terminal command initiated from an EXEC as a CEMT request	The command can be issued using the MVS command facility (CMD) if the console used is defined to CICS in the CICS Terminal Control Table.

Example

This example command switches dump data sets.

```
/* REXX */
"IMFEXEC CICS CEMT SET DUMP SWI"
```

. or

```
/* REXX */
"IMFEXEC CICS CEMTQ SET DUMP SWI"
```

CICS CHAP

This command causes a dynamic change to the priority of an active task.

Command	Parameters
CICS CHAP	Taskno Priority

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
Taskno	Number of the currently active task to modify	Decimal numeric value in CICS allowable range. This number can be obtained by using the IMFEXEC QUERY command (this requires MAINVIEW for CICS to be installed). Looping transactions running at a dispatching priority of 255 sometimes cannot be changed.
New priority	The priority to assign to this task	Numeric value in the range 0-255.

Example

This example command assigns a dispatching priority of 232 to task 8756.

```
/* REXX */
"IMFEXEC CICS CHAP 8756 232"
```

CICS CICSKEY

This command changes CICSKEY settings for CICS transactions.

Command	Parameters
CICS CICSKEY	Tran ID .br;[YES|NO]

The following table describes the parameters.

Parameter	Function	Notes
Tran ID	The name of a CICS transaction	
[YES|NO]	Can be set to YES or NO	

Example

This section contains examples using the IMFEXEC CICS CICSKEY command statement. A brief discussion follows each example.

Example 1

```
"IMFEXEC CICS CICSKEY CEMT YES"
```

This example command sets the TASKDATAKEY of the CICS CEMT to CICS.

Example 2

```
"IMFEXEC CICS CICSKEY CEMT NO"
```

This example command sets the TASKDATAKEY of the CICS CEMT to USER.

CICS CLOSE

This command closes one file in the CICS region.

Command	Parameters
CICS CLOSE	Filename

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
Filename	The filename of the file to close	1- to 8-alphanumeric file name defined in the CICS File Control Table (FCT).

Example

This example command closes a CICS file.

```
/* REXX */
"IMFEXEC CICS CLOSE P001"
```

CICS CONN

This command alters the status of IRC/ISC connections.

Command	Parameters
CICS CONN	SYSID IN OUT ACQ REL NOTPEND PURGE

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
SYSID	Is the CICS SYSID for the MRO/ISC connection	Values can be: **INservice** Puts the connection into service **OUTservice** Takes the connection out of service **ACQuire** Acquires a connection **RELease** Releases a connection **NOTPEND** Makes a connection not pending **PURGE** Purges a connection

Example

This example command puts a connection in service.

```
"IMFEXEC CICS CONN SYSID IN"
```

CICS DISABLE

This command command makes a resource unavailable to applications, except for those currently using it.

Command	Parameters
CICS DISABLE	FILE|TRAN|PROGRAM|DEST Identifier

The following table describes the parameters.

Parameter	Function	Notes
Type	The type of resource to affect	Values are: **FILE** A file **Note:** BBI-SS PAS to CICS communication must be active. **TRAN** A CICS transaction **PROGRAM** A CICS application program **DEST** A transient data queue
Identifier	The resource ID for each type	Values are: **file id** Identifier is a 1- to 8-alphanumeric file name **tran id** Identifier is a 1- to 4-alphanumeric transaction name **program id** Identifier is a 1- to 8-alphanumeric program name **dest id** Identifier is a 1- to 4-character queue name defined in the Destination Control Table (DCT)

Example

This example command disables a CICS transaction.

```
/* REXX */
"IMFEXEC CICS DISABLE TRAN ABRW"
```

CICS DROP

This command decreases the use-count of a program.

Command	Parameters
CICS DROP	Program name

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
Identifier	A CICS program identifier	1- to 8-character ID of the program affected. After a program use-count reaches 0, it is eligible to be removed by CICS program compression. You should be careful to avoid dropping, and potentially removing, programs that are actually in use by executing transactions.

Example

This example command decreases the use-count of the program DSPFILE.

```
/* REXX */
"IMFEXEC CICS DROP DSPFILE"
```

CICS DUMPDB

This command prepares a database for dumping by preventing updates so a backup job can be run in another region.

Command	Parameters
CICS DUMPDB	Database name

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
Database	Database identified in the Data Management Block Directory (DMB)	1- to 8-character name of the database.

Example

This example command prepares the database STD2XCP for batch updates.

```
/* REXX */
"IMFEXEC CICS DUMPDB STDCX2P"
```

CICS ENABLE

This command makes a resource available for use.

Command	Parameters
CICS ENABLE	FILE|TRAN|PROGRAM|DEST Identifier

The following table describes the parameters.

Parameter	Function	Notes
Type	The type of resource to affect	Values are: **FILE** A file **Note:** BBI-SS PAS to CICS communication must be active. **TRAN** A CICS transaction **PROGRAM** A CICS application program **DEST** A transient data queue
Identifier	The resource ID for each type	Values are: **file id** Identifier is a 1- to 8-alphanumeric file name **tran id** Identifier is a 1- to 4-alphanumeric transaction name **program id** Identifier is a 1- to 8-alphanumeric program name **dest id** Identifier is a 1- to 4-character queue name defined in the Destination

Example

This example command enables the CICS transaction ABRW.

```
/* REXX */
"IMFEXEC CICS ENABLE TRAN ABRW"
```

CICS FREE

This command deallocates a file from the CICS region or BBI-SS PAS.

Command	Parameters
CICS FREE	Filename [LOCAL]

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
Filename	The DD Name of the file to deallocate	1- to 8 alphanumeric. DISABLE FILE and CLOSE commands for the data set must be issued before FREE. The name does not need to be one that is specified in the CICS FCT, but the file must be closed to be freed.
LOCAL	Forces deallocation from the BBI-SS PAS instead of the CICS region	

Example

These example commands close and deallocate a data set.

```
/* REXX */
"IMFEXEC CICS CLOSE MASTER"
"IMFEXEC CICS FREE  MASTER"
```

CICS INSERVE

This command puts a terminal, line, or control unit in service.

Command	Parameters
CICS INSERVE	TERMINAL|LINE|CONTROLLER Identifier

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
Type	The type of resource to modify	One of the following: TERMINAL LINE CONTROLLER If you use generics for the terminal ID name, a maximum of only 200 discrete commands are executed.
Identifier	ID of the terminal, line, or controller	1-4 characters. The ID of a line or a controller cannot be specified as a generic.

Example

This example command makes all terminals with IDs that begin with SC available for use.

```
/* REXX */
"IMFEXEC CICS INSERVE TERMINAL SC*"
```

CICS ISOLATE

This command changes ISOLATE settings for CICS transactions.

Command	Parameters
CICS ISOLATE	Tran ID [YES|NO]

The following table describes the parameters.

Parameter	Function	Notes
Tran ID	The name of a CICS transaction	
[YES|NO]	Can be set to YES or NO	

Example

This section contains examples using the IMFEXEC CICS ISOLATE command statement. A brief discussion follows each example.

Example 1

```
"IMFEXEC CICS ISOLATE CEMT YES"
```

This example command sets CICS CEMT to ISOLATE(YES).

Example 2

```
"IMFEXEC CICS ISOLATE CEMT NO"
```

This example command sets CICS CEMT to ISOLATE(no).

CICS KILL

This command terminates a CICS task identified by a CICS task number or identified by the CICS terminal it is attached to.

Note: When this command is used on a task running in a CICS/ESA region, the task's system purgeable mask is turned on (SPURGE set to YES) prior to execution of the command.

Command	Parameters
CICS KILL	TASK Task number [WITH DUMP] [FORCE|PURGE|FORCEPURGE]
	TERMINAL Terminal ID [PURGE|FORCEPURGE]

The table describing the IMFEXEC CICS KILL TASK command statement parameters is on page 288 and the table describing the IMFEXEC CICS KILL TERM command statement parameters is on page 290.

CICS KILL TASK

The following table describes the parameters for the IMFEXEC CICS KILL TASK command statement.

Parameter	Function	Notes
Task number	The number of the task affected	A CICS-assigned task number from 1 to 99999.

Parameter	Function	Notes
WITH	Readability token	The WITH parameter works only with DUMP.
Type	Type of abnormal termination desired	One of the following: **DUMP** If the integrity of the CICS region can be maintained, the task is abnormally ended with a dump. **FORCE** Forces a looping task to abend with a dump, regardless of integrity exposure. Expect to use this service more than once on a multiprocessor for a task in a loop. CAUTION: This can cause the CICS region to abend. **PURGE** Purges a task using the services of the CICS supplied transaction CEMT. **FORCEPURGE** PURGE a task using the services of the CICS supplied transaction CEMT using the FORCE parameter. **Note:** BBI-SS PAS to CICS communication **must** be active to use the PURGE and FORCEPURGE parameters. If only the task number is specified, the task is abnormally terminated if the integrity if the CICS region can be maintained. A dump is produced.

CICS KILL TERM

The following table describes the parameters for the IMFEXEC CICS KILL TERM command statement.

Note: BBI-SS PAS to CICS communication **must** be active.

Parameter	Function	Notes
Terminal ID	The terminal that the task is attached to	
Type	Type of abnormal termination desired	One of the following: **PURGE** Purges a task using the services of the CICS supplied transaction CEMT. **FORCEPURGE** PURGE a task using the services of the CICS supplied transaction CEMT using the FORCE parameter.

Examples

This section contains an example using the IMFEXEC CICS KILL TASK and IMFEXEC CICS KILL TERM command statements. A brief discussion follows the example.

Example 1 - IMFEXEC CICS KILL TASK

```
/* REXX */
"IMFEXEC CICS KILL TASK 0004"
```

When coded within an EXEC driven off the message FT041S, this command kills a task when the message:

```
FT04IS TRAN xxx TASK yyyy USING zzzK BYTES
```

is logged to the online Journal. The task ID is contained in the P004 variable.

Example 2 - IMFEXEC CICS KILL TERM

```
/* REXX */
"IMFEXEC CICS KILL TERM BSA4"
```

This example terminates the CICS task attached to terminal BSA4.

CICS LOAD

This command increases the use-count of a program.

Command	Parameters
CICS LOAD	Program name

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
Identifier	The name of the affected program	1- to 8-character alphanumeric. If a program is not currently resident in CICS storage, it will be loaded. Unless the use count is specifically decreased with the DROP transaction or through CICS services, the program stays permanently loaded until CICS terminates.

Example

This example command increases the use count of the program named PROGX470.

```
/* REXX */
"IMFEXEC CICS LOAD PROGX470"
```

CICS NEWCOPY

This command marks the program name in the PPT nonresident and refreshes its disk address to prepare for a newly link-edited version or restoration of that program.

Command	Parameters
CICS NEWCOPY	Program name

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
Program	The program to refresh	1- to 8 alphanumeric.

Example

This example command refreshes the CICS copy of a program named PGM1.

```
/* REXX */
"IMFEXEC CICS NEWCOPY PGM1"
```

CICS OPEN

This command opens a file in the CICS region.

Command	Parameters
CICS OPEN	File name

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
Filename	The name of the FCT entry to open	1- to 8 characters alphanumeric file name.

Example

These example commands allocate and open a data set.

```
/* REXX */
"IMFEXEC CICS ALLOC MAIN1 TO USERV.MAIN1.CLUSTER"
"IMFEXEC CICS OPEN MAIN1"
```

CICS OUTSERVE

This command takes a terminal, line, or control unit out of service.

Command	Parameters
CICS OUTSERVE	TERMINAL|LINE|CONTROLLER Identifier

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
Type	The type of resource to modify	One of the following: TERMINAL LINE CONTROLLER If you use generics for the terminal ID name, a maximum of only 200 discrete commands are executed.
Identifier	ID of the terminal, line, or controller	1-4 characters. The ID of a line or a controller cannot be specified as a generic.

Example

This example command keeps all terminals with IDs that begin with T and end with S from being used.

```
/* REXX */
"IMFEXEC CICS OUTSERVE TERMINAL T++S"
```

CICS PURGE

This command terminates a CICS resource

Command	Parameters
CICS PURGE	TSUT|ICE|DEST|AID

BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes

TSUT value or value HEX	Purges a temporary storage unit from the CICS system.	Can be up to 16 characters or a 32 character representation of a 16-byte hexadecimal number.
		
		The maximum depends on the CICS release.
		
		Example 1:
		
		`"IMFEXEC CICS PURGE TSUT PAYROLL1"`
		
		Example 2:
		
		`"IMFEXEC CICS PURGE TSUT 1C3A773B HEX"`
		`/* Purge the binary TSUT with binary ID 1C3A773B */`

ICE value	Purges an interval control element from the CICS system.	Can be up to 8 characters or a 16 character representation of an 8-byte hexadecimal number.
		
		Example 1:
		
		`"IMFEXEC CICS PURGE ICE DELAY"`
		
		Example 2:
		
		`"IMFEXEC CICS PURGE ICE 3C0000FF00001000”`
		`/* Purge the binary ICE with binary ID 3C0000FF00001000 */`

Parameter	Function	Notes
DEST value	Deletes the CICS Transient Data queue.	Up to 4 character queue name allowed. Example: `"IMFEXEC CICS PURGE DEST DEVL" /* Delete the development queue */`
AID value termed	Purges an Automatic Initiation Descriptor from the CICS system.	Can be up to 8 characters or a 16 character representation of an 8-byte hexadecimal number. Example: `"IMFEXEC CICS PURGE AID 3C0000FF00001000 L287 TRN1" /* Purge the AID for terminal L287 transaction TRN1 */`

Example

Examples are located in the Parameters table with the description of each keyword.

CICS QUERY

This command invokes MAINVIEW for CICS interactive services, such as SUMMARY, MONITOR, PROBLEM, SUBPOOL, and so on.

Command	Parameters
CICS QUERY	Command

QUERY is executed on behalf of the target CICS system. When QUERY is used, data normally sent to the terminal is written into variables that the EXEC can analyze. The output returned from the command is written into the LOCAL variables LINE1 through LINExx, which correspond to the lines of the screen image.

The variable IMFNOL contains the number of lines of output. These variables need to be retrieved using the VGET statement before they can be used.

The output is similar to output produced through the same service of a MAINVIEW for CICS terminal session, but this output does not have screen attributes in the variables. The data returned also does not contain the first two lines displayed when invoking the service under the BBI-TS. The data returned does not contain the header lines which are displayed when invoking the service under the BBI-TS unless they contain variable data.

You might need to experiment with the correct offsets to use when substringing particular items. Changes to MAINVIEW for CICS display formats might affect EXECs that process this data.

For most MAINVIEW for CICS services, two header lines are omitted before the data is returned in the LOCAL variables. There are some exceptions to this standard.

One exception is those services that contain variable data in one of the header lines. For these services, the header line containing the variable data and all subsequent lines are returned.

Another exception is those services which contain either one or two blank header lines. For these services, the header lines are not returned at all. However, all blanks that might be interspersed within the detail lines of a specific display are passed to the EXEC. You must accommodate for these lines in the EXEC.

The following table describes the parameters.

Parameter	Function	Notes
Command	A MAINVIEW for CICS service request, referred to in format descriptions in the *MAINVIEW for CICS PERFORMANCE MANAGER User Guide*	No quotes are required around this operand.

Example

This section contains an example using the IMFEXEC CICS QUERY command statement. A brief discussion follows the example.

```
/* REXX */
"IMFEXEC CICS QUERY SUBPOOL"
/* DSA PERCENTAGE IS NOW IN MSG #2, COLUMNS 14-16 */
"IMFEXEC VGET LINE2 LOCAL"
DSAPERC = SUBSTR(LINE2,14,2)
IF DSAPERC < 50 THEN CALL LOKAY
```

The above example command shows an EXEC that interprets the DSA utilization percentage. This could be used to influence decisions made within an EXEC that is invoked when message FT041S (task using excessive storage) is issued.

To page through several MAINVIEW for CICS displays, supply a parameter to the second and subsequent information of a command. This parameter should specify the last item on the previous page; for example, when using the TRAN display, invoke the TRAN display on the second iteration with the name of the last transaction displayed on the panel you have already processed (IMFEXEC QUERY TRAN xxxx). MAINVIEW for CICS begins the next display with that transaction.

CICS RECOVERDB

This command prepares a database for recovery by preventing reads and updates so a recovery utility can be run in another region.

Command	Parameters
CICS RECOVERDB	Database name

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
Database	The name of the database identified in the Data Management Block Directory (DMB)	1- to 8-characters alphanumeric.

Example

This example command inhibits online updates to the database STDIDBP.

```
/* REXX */
"IMFEXEC CICS RECOVERDB STDIDBP"
```

CICS RELEASE

This command releases VTAM terminals from CICS.

Command	Parameters
CICS RELEASE	TERMINAL Terminal ID

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
Terminal identifier	ID of the terminal to be released	1- to 4-characters alphanumeric. If you use generics for the terminal ID name, a maximum of only 200 discrete commands are executed.

Example

This example command releases all terminals beginning with LM.

```
/* REXX */
"IMFEXEC CICS RELEASE TERMINAL LM*"
```

CICS SPURGE

This command dynamically changes the SPURGE value for a CICS transaction.

Command	Parameters
CICS SPURGE	Tranid [YES|NO]

Note: For CICS/ESA, the stall purge mechanism no longer exists. SPURGE now indicates whether a transaction is system purgeable. If the transaction definition specifies SPURGE=NO, the transaction is protected from deadlock timeout purge and purge requests (but not from force purge requests) issued by applications or the master terminal.

The following table describes the parameters.

Parameter	Function	Notes
Tranid	Transaction to affect	
Status	YES/NO	Specifying YES turns the SPURGE flag on. Specifying NO turns the SPURGE flag off.

Example

This example command sets the SPURGE flag for transaction RT17 to on.

```
/* REXX */
"IMFEXEC CICS SPURGE RT17 YES"
```

CICS STARTDB

This command activates a database, making it available for processing.

Command	Parameters
CICS STARTDB	Database name

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
Database	Database identified in the Data Management Block Directory (DMB)	1- to 8-character name of the database.

Example

This example command activates a database with the name STDCDBP.

```
/* REXX */
"IMFEXEC CICS STARTDB STDCDBP"
```

CICS STOPDB

This command deactivates a database, making it unavailable for processing.

Command	Parameters
CICS STOPDB	Database name

Note: BBI-SS PAS to CICS communication must be active.

The following table describes the parameters.

Parameter	Function	Notes
Database	Database identified in the Data Management Block Directory (DMB)	1- to 8-character name of the database.

Example

This example deactivates the database STDCX2P.

```
/* REXX */
"IMFEXEC CICS STOPDB STDCX2P"
```

CICSTRAN

This command invokes a CICS transaction.

Command	Parameters
CICSTRAN	Tran ['Parameters']

The following table describes the parameters.

Parameter	Function	Notes
Tran	The ID of the transaction to invoke	1- to 4-alphanumeric characters. The transaction must be capable of running the terminal unattached. For example, use EXEC CICS RETRIEVE instead of EXEC CICS RECEIVE.
Parameters	Any parameters necessary for this transaction	Maximum length is 80 characters.

Example

This example command deactivates BBI-SS PAS to CICS communications.

```
/* REXX */
"IMFEXEC CICSTRAN FST2 'QOFF'"
```

CMD

The IMFEXEC CMD command performs a variety of functions depending on the parameters supplied. It may be used to:

- Issue a BBI control command
- Issue an MVS command
- Issue an IMS command
- Issue a JES3 command

Command types are recognized by their command characters (the first character of the command). A missing or invalid command character causes the command to be issued and treated like an MVS command.

There are two major command formats: commands that return a full response and those that do not (or do so in a limited fashion). In general, enclosing the command argument in quotes indicates that a response should be returned to the EXEC.

The different versions of IMFEXEC CMD are:

- CMD - Issue BBI command without response, page 306
- CMD - Issue BBI command with response, page 307
- CMD - Issue MVS command with response (and with X-MCS consoles), page 310
- CMD - Issue IMS command without response, page 315
- CMD - Issue IMS command with response, page 317

CMD (Issue BBI Command without Response)

This command issues a BBI control command.

Command	Parameters
CMD	.Command [p1 ... pn]

The following table describes the parameters.

Parameter	Function	Notes
Command and parameters	The command or command abbreviation and any parameters	A period (.) identifies the command as a BBI control command. See the *MAINVIEW Common Customization Guide* for a full description of the BBI control commands.

Condition codes are listed in the following table.

Value	Description
0	This command format always returns a zero condition code.

Example

This example command switches the active BBI-SS PAS Journal log data set to an alternate data set. The command and any response are written to the BBI-SS PAS Journal log, which is viewed from LOG DISPLAY.

```
/* REXX */
"IMFEXEC CMD .I JOURNAL"
```

CMD (Issue BBI Command with Response)

This command format issues BBI commands. A response is returned to the issuing EXEC.

Command	Parameters
CMD	'.Command [p1 ... pn]' TYPE(BBI) ALL ALLWAIT(1 - 9999)

Command output is placed in LOCAL variables LINE1 through LINEnnnn, where nnnn is the last line (variable IMFNOL contains the value of nnnn).

The following table describes the parameters.

Parameter	Function	Notes
.Command and parameters	The command or command abbreviation and any parameters	The period (.) identifies the command as a BBI control command. See the *MAINVIEW Common Customization Guide* for a full description of the BBI control commands.
TYPE	Command response designator	Must be BBI. If this is not specified, the command will be issued as an MVS command.
ALL	Retrieve all responses	This parameter causes all other criteria to be ignored and to wait for further responses as long as responses continue to arrive within half-second intervals.

Parameter	Function	Notes
ALLWAIT	Specify an interval to wait from 1 to 9999 seconds.	ALLWAIT allows CMD processing to continue waiting in intervals (specified in seconds) until no responses are received within an interval of that length. If at least one response is received in that interval, processing continues for an additional interval. This processing is repeated until no responses are received within an interval, which may result in added wait time. Therefore small intervals of 1-5 are recommended. Example of processing: Sample command: `IMFEXEC CMD 'cmd_text' TYPE(BBI) ALL ALLWAIT(3)` Processing waits 3 seconds as specified in ALLWAIT and then checks to see if any responses were received. If none, the command is terminated. If a response was received, processing waits an additional 3 seconds and checks again. This action is repeated until no responses are recieved within the specified interval. ALLWAIT is only valid when ALL is specified.

Condition codes are listed in the following table.

Value	Description
0	Command response returned before WAIT time expired
4	Command partially returned after WAIT time expires
8	No reply has been received

Example

This example EXEC sends the BBI .D A command output to a TSO user ID.

```
/* REXX */
"IMFEXEC CMD '.D A' TYPE(BBI) ALL"
IF IMFCC < 5 THEN DO
  DO N = 1 TO IMFNOL
  "IMFEXEC VGET LINE"N "LOCAL"
  "IMFEXEC SEND  'LINE"N VALUE('LINE'N) "' USER(BBI1)"
  END
END
```

CMD (MVS Version with Response through X-MCS Consoles)

This command issues an MVS command using X-MCS consoles. A response is returned to the issuing EXEC.

Command	Parameters
CMD	'#Command' [RESPONSE(*|Message ID)] [COUNT|LINES(1|n)] [WAIT(30|n)] [CONSOLE(n)|NAME(xxxxxxxx)] [ALL] [ALLWAIT(1 - 9999)] [MIGID(yes|no)] [DEBUG]

The console choice is automatic and transparent. Command output is placed in LOCAL variables LINE1 through LINEnnnn, where nnnn is the last line (variable IMFNOL contains the value of nnnn). These variables must be retrieved using the VGET command before they can be used.

The number of X-MCS consoles allocated controls the number of commands that can be processed concurrently. If all consoles are being used and an EXEC issues a command, the EXEC waits until another EXEC releases the console.

In addition, prior to MVS Version 4, all consoles have only a 1-byte console ID. Beginning with MVS Version 4, all consoles (subsystem, MCS, and X-MCS) have a 4-byte console ID and an 8-byte console name. AutoOPERATOR creates all X-MCS console names using the format:

`SSIDnnnn`

where:

SSID Is the BBI-SS PAS identifier name

nnnn Is a number from 0 to the total number of X-MCS consoles created

Note: You must make sure that no other application uses these console names.

Some consoles may have a 1-byte console ID in addition to the new 4-byte console ID. For example, MCS consoles (defined in the CONSOLxx member of SYS1.PARMLIB) continue to have a 1-byte console ID in addition to the 4-byte console ID and 8-byte console name. However, X-MCS consoles usually do not have a 1-byte console ID.

This means that applications that interface with consoles specified in the CONSOLxx member of SYS1.PARMLIB do not have to be updated to understand 4-byte console IDs.

Applications that will interface with X-MCS consoles need to be updated to understand 4-byte console IDs. You may have some applications **that do not yet understand 4-byte console IDs**. To remain compatible with these applications, MVS allows some X-MCS consoles to have a 1-byte migration ID (MIGID) specified. Therefore, X-MCS consoles that have a MIGID can interface with applications that have not yet been updated.

In addition, within a sysplex, MVS limits the number of X-MCS consoles with MIGIDs. For this reason, AutoOPERATOR does not request a MIGID for all X-MCS consoles it creates. Therefore, you must determine which EXECs using the IMFEXEC CMD statement will need to specify a MIGID.

Refer to the *MAINVIEW AutoOPERATOR Customization Guide* for more information about MVS console considerations and how X-MCS consoles are allocated with and without MIGIDs.

Command	Parameters	Notes
'#Command'	MVS command to be issued	The maximum length of an MVS command is 126 characters. To prevent the BBI-SS PAS from interpreting the MVS command command as a BBI command, make sure you prefix the MVS command with a pound sign (#). Prefixing the command with a # causes AutoOPERATOR to treat the command as an MVS command. The # is stripped off the command before it is issued. If the command you want to issue begins with a #, make sure you prefix the command with 2 pound signs: '##command'.
RESPONSE	Message ID(s) expected for response	The default is '*', which means any message. You can specify up to 8 message IDs, separated by commas, each up to 16 characters long. Wildcards are allowed. If RESPONSE(*) is specified, the EXEC picks up all messages from the selected MVS console. If there are messages that are responses to previous commands on the same MVS console, it is recommended that RESPONSE is coded for the MSG ID.
COUNT|LINES	Number of response lines to be retrieved	Default is 1. You may specify from 0 through 9999. A Multi Line WTO (MLWTO) is counted as one line (even though it may be composed of many lines, as in some VTAM command responses). If COUNT(0) is explicitly coded, it means no response is needed. This format is recommended over using the IMFEXEC CMD without response statement.
WAIT	Length of time to wait for all response lines to arrive	Default is 30 seconds. You may specify from 5 through 999 seconds.
CONSOLE	A 1-byte console ID to issue the command from	This is needed only under unusual conditions and you must have a valid, active MVS console available or no response can be obtained.
ALL	Retrieve all responses	This parameter causes all other criteria to be ignored and to wait for further responses as long as responses continue to arrive within half-second intervals.

Command	Parameters	Notes
ALLWAIT	Specify an interval to wait from 1 to 9999 seconds.	ALLWAIT allows CMD processing to continue waiting in intervals (specified in seconds) until no responses are received within an interval of that length. If at least one response is received in that interval, processing continues for an additional interval. This processing is repeated until no responses are received within an interval, which may result in added wait time. Therefore small intervals of 1-5 are recommended. Example of processing: Sample command: `IMFEXEC CMD 'cmd_text' TYPE(BBI) ALL ALLWAIT(3)` Processing waits 3 seconds as specified in ALLWAIT and then checks to see if any responses were received. If none, the command is terminated. If a response was received, processing waits an additional 3 seconds and checks again. This action is repeated until no responses are recieved within the specified interval. ALLWAIT is only valid when ALL is specified.

Command	Parameters	Notes
NAME	A valid MVS console name	Use this parameter if the command must be issued from a specific MVS console name. When command responses are expected, one of three things can happen (depending on the state of the console of the console name you specified): • An active console identified to AutoOPERATOR will be used. • No X-MCS consoles are defined so one will be created and it will not have a MIGID. When the command ends, the console is deactivated. • An inactive X-MCS console is activated and used for the command. • When the command ends, the console is deactivated. If command responses are not required (COUNT=0), any valid MVS console (defined or undefined, active or inactive) may be specified. The NAME and CONSOLE parameters cannot be used together.
MIGID	Specify YES or NO to use an X-MCS console with a MIGID.	MAINVIEW AutoOPERATOR default is NO. If you specify YES, an X-MCS console with a MIGID is used.
DEBUG	Issues debugging messages	Used for problem diagnosis.

The IMFEXEC CMD with response results in some variables being set in addition to IMFCC. IMFCCON contains the 1-byte console ID or migration ID (decimal). If the Extended MCS console does not have a migration ID, the variable IMFCCON contains 255. The variable IMFCNAME contains the console name.

The variable IMFRC contains the return code given by the MVS MGCRE macro (which is used to issue the command). This return code is meaningful only when issuing the MVS START command. Do not inspect this variable if you are not issuing the MVS START command. When you issue the MVS START command and if IMFRC is zero, the variable IMFCASID contains the ASID (decimal) of the started address space and IMFCSTKN contains the STOKEN (16 hexadecimal characters).

Value	Description
0	Command responded within WAIT time
4	Command partially responded within WAIT time
8	No reply has been received, WAIT time has expired

Value	Description
16	Command text is greater than 121 characters
20	Severe error: see short message text for more information

Examples

This section contains two examples using the IMFEXEC CMD command statement. A brief discussion follows each example.

Example 1

```
/* REXX */
PARSE ARG EXNAME .
"IMFEXEC MSG '."EXNAME "EID="IMFEID"'"
"IMFEXEC CMD '#D NET,CDRMS' RESPONSE(IST350I)"
"IMFEXEC MSG '."EXNAME "IMFNOL="IMFNOL "CC="IMFCC"'"

DO I=1 TO IMFNOL
  "IMFEXEC VGET LINE"I "LOCAL"
  "IMFEXEC MSG '."EXNAME "LINE"I "LENGTH="LENGTH(VALUE('LINE'I))"'"
  "IMFEXEC MSG '."EXNAME VALUE('LINE'I)"'"
END

"IMFEXEC MSG '."EXNAME "EID="IMFEID "ENDED'"
```

This EXEC demonstrates how to issue a VTAM command with response. Note, VTAM typically returns its responses as a Multi-line WTO (MLWTO); therefore, the COUNT parameter should be set to one (the default).

Example 2

```
/* REXX */
PARSE ARG EXNAME .
"IMFEXEC MSG '."EXNAME "EID="IMFEID"'"
"IMFEXEC CMD '#$DJ1-999,L=Z' RESPONSE($HASP636)"
"IMFEXEC MSG '."EXNAME "IMFNOL="IMFNOL "CC="IMFCC"'"

DO I=1 TO IMFNOL
  "IMFEXEC VGET LINE"I "LOCAL"
  "IMFEXEC MSG '."EXNAME "LINE"I "LENGTH="LENGTH(VALUE('LINE'I))"'"
  "IMFEXEC MSG '."EXNAME VALUE('LINE'I)"'"
END

"IMFEXEC MSG '."EXNAME "EID="IMFEID "ENDED'"
```

This REXX EXEC demonstrates how to issue a JES2 command with response. Note that requesting JES2 to return its responses as a Multi-line WTO (MLWTO) through the L=Z option provides a more reliable means to make sure that you receive all the response lines. Since JES2 (in this case) returns one MLWTO ($HASP636, even though it comprises many lines), the COUNT parameter should be set to one (the default).

CMD (Issue IMS Command without Response)

This command format issues IMS commands. Only minimal response is returned.

Generic resource names can be specified in the commands using wildcard characters. The plus sign (+) can be used to represent any one character, while the asterisk (*) can be used to represent any number of characters.

Command	Parameters
CMD	/IMS command

The response segment is returned in the standard CLIST variable SYSDVAL, which can be parsed using the READDVAL command. READDVAL functions the same way as in a TSO CLIST.

Note: Command response is not returned when the /MODIFY or /MSVERIFY command is issued.

Parameters	Function
/IMS command	The IMS command to be issued

Condition codes are listed in the following table.

Value	Description
0	Command issued and first segment of response returned in SYSDVAL.
4	Generic command format resulted in multiple IMS commands. SYSDVAL contains response to first command.
8	Command timeout, no response returned (Msg IM9215W issued).
12	One of the following: • Target IMS not available • The message, `IO1317W Command Not Issued, No Matching Resource Found` is returned as a response when there are no matching resources found.

Examples

This section contains examples using the IMFEXEC CMD command statement. A brief discussion follows each example.

Example 1 - Issuing generic commands

```
/* REXX */
"IMFEXEC CMD /STA DATABASE BE3ORDER"
"IMFEXEC CMD /STA DATABASE BE3*"
"IMFEXEC CMD /STA DATABASE +++ORDER"
"IMFEXEC CMD /STA DATABASE BE+ORDER"
```

In this example, AutoOPERATOR issues generic /STA DATABASE commands to start all databases whose names begin with BE3 or contain the characters ORDER in positions 4-8.

The * cannot be followed by any other characters and only one can be used in a string. You can use a + and an * together in a generic IMS resource command but the * must be the last character.

Example 2 - Retrieving &SYSDVAL

```
/* REXX */
"IMFEXEC CMD /STA TRAN TE4COCNG"
/* SYSDVAL = DFS058 COMMAND COMPLETED EXCEPT FOR TE4COCNG */
READDVAL MSGID P1 P2 P3 P4 P5
IF P3 = 'EXCEPT' THEN DO
  commands
END
```

Starts an IMS transaction and verifies that the start command worked. This method of issuing an IMS command (no quotation marks) returns only the first response segment to the EXEC. Additional response segments are not available to the EXEC. See the description of IMS command with response in the next section for information about accessing all response segments in an EXEC.

CMD (Issue IMS Command with Response)

This command format issues IMS commands. A response is returned to the issuing EXEC.

Generic resource names can be specified in the commands using wildcard characters. The plus sign (+) can be used to represent any one character, while the asterisk (*) can be used to represent any number of characters. Only one by any other characters. The + and in a string.

Note: Generic names are not supported for the ?RMxxxxxx DBRC (Database Recovery Control) commands.

Command	Parameters
CMD	'/IMS command' [COUNT(1|n)] TYPE(IMS) DBCTL(dbctltgt) [WAIT(30|n)] ALL ALLWAIT(1 - 9999)

The response segment is returned in the local variable pool in variable LINE1 through LINEnnnn. The number of lines returned is available in IMFNOL.

Note: Command response is not returned when the /MODIFY or /MSVERIFY command is issued.

Parameters	Function	Notes
'/IMS or DBCTL command'	The command to be issued	The maximum length of the IMS command is 252 bytes. **Note:** The / (slash) designates this command format as an IMS or DBCTL command. The quotes indicate that a response is to be returned.
COUNT	The maximum number of response segments	Numeric value in the range 1-9999. This parameter is required. When the response to a command is an IMS multi-segment message, the IMFEXEC CMD TYPE(IMS) stops waiting when any of the following conditions is met: • WAIT time has expired. • COUNT value has been met. • IMS sent the last segment of a multi-segment message.
TYPE	Command response designator	Must be IMS. This parameter is required. If this is not specified, the command will be issued as an MVS command.

Parameters	Function	Notes
DBCTL	DBCTL target address space name	Must be used for DBCTL-only address spaces. Must not be used for IMS and DBCTL address spaces.
WAIT	The maximum amount of time, in seconds, to wait for a command response	Numeric value in the range 5-9999. When the response to a command is an IMS multi-segment message, the IMFEXEC CMD TYPE(IMS) stops waiting when any of the following conditions is met: • WAIT time has expired. • COUNT value has been met. • IMS sent the last segment of a multi-segment message.
ALL	Retrieve all responses	This parameter causes all other criteria to be ignored and to wait for further responses as long as responses continue to arrive within half-second intervals.
ALLWAIT	Specify an interval to wait from 1 to 9999 seconds.	ALLWAIT allows CMD processing to continue waiting in intervals (specified in seconds) until no responses are received within an interval of that length. If at least one response is received in that interval, processing continues for an additional interval. This processing is repeated until no responses are received within an interval, which may result in added wait time. Therefore small intervals of 1-5 are recommended. Example of processing: Sample command: `IMFEXEC CMD 'cmd_text' TYPE(BBI) ALL ALLWAIT(3)IMFEXEC CMD` Processing waits 3 seconds as specified in ALLWAIT and then checks to see if any responses were received. If none, the command is terminated. If a response was received, processing waits an additional 3 seconds and checks again. This action is repeated until no responses are received within the specified interval. ALLWAIT is only valid when ALL is specified.

IMS commands are issued by way of the IMS internal interface, which returns the response to the issuing EXEC. DBCTL commands are issued as MVS commands prefixed with the DBCTL command character, and MVS returns the response.

Condition codes are listed in the following table.

Value	Description
0	Command responded within WAIT time
4	Command partially responded within WAIT time
8	No reply has been received, WAIT time has expired
12	Target IMS not active; no matching resource for generic command

Examples

This section contains examples using the IMFEXEC CMD command statement.

Example 1

```
   /* REXX */
    "IMFEXEC CMD '/DIS TRAN TH*' COUNT(20) TYPE(IMS)"
    DO N = 1 to IMFNOL
      "IMFEXEC VGET LINE"N" LOCAL"
      "IMFEXEC MSG 'LINE"N"="VALUE('LINE'N)" ' "
    END
EXIT
```

This example requests the transactions whose trancodes begins with TH and receives up to 20 response segments.

Example 2

```
/* REXX */
"IMFEXEC CMD '/DIS TRAN TH*' COUNT(20) TYPE(IMS)"
"IMFEXEC VDCL DISPLAY LIST (MSG A B C D E F G H I J K L M)"
KEY = TIME('S')
    DO N = 1 TO IMFNOL
       "IMFEXEC VGET LINE"N" INTO (DISPLAY) LOCAL"
        IF (MSG = T02) & (F > 0) THEN DO
             "ALERT "KEY" ' 'TRAN = "A" COUNT = "F" ' FUNCTION(ADD)",
             " QUEUE(TEST) ORIGIN(REGIS) COLOR(RED)"
         END
    END

EXIT
```

This example utilizes the VDCL function to obtain the first 20 transactions starting with TH* and create Alerts for those with non-zero PLCT. These Alerts will be created with a key = current time in seconds and the text will include the transaction name and the PLCT value.

Example 3

```
/*REXX*/
"IMFEXEC CMD '/DIS A' ALL TYPE(IMS)"
"IMFEXEC MSG 'IMFCC="IMFCC" IMFNOL="IMFNOL"'"
DO I = 1 TO IMFNOL
"IMFEXEC VGET LINE"I" LOCAL"
"IMFEXEC MSG 'LINE"I"="VALUE('LINE'I)" ' "
END
EXIT
```

This example shows how to issue the IMS command /DIS A with a response and process the LINE1...LINEI variables, which contain the response messages.

CNTL

This command controls the general processing flow and characteristics of an EXEC.

Command	Parameters
CNTL	[CMD|NOCMD] [LIST|NOLIST] [PERLIM()] [TIMLIM()] [SELLIM()] [MAXTPUT()] [GLOBAL|LOCAL]

Use this command for help in debugging AutoOPERATOR EXECs. CNTL LIST causes all commands issued with the EXEC to be listed in the BBI-SS PA Journal log. Refer to Chapter 13, “Testing and Debugging EXECs Interactively” on page 411 for more information about testing EXECs.

The following table describes the parameters.

Parameter	Function	Notes
CMD	Normal EXEC processing is in effect	Default
LIST	List every EXEC command in the BBI-SS PAS log	
NOCMD	Do not process action commands	When NOCMD is in effect in the current CLIST, the following IMFEXEC commands are not executed: CMD, CICSTRAN, IMSTRAN, SUBMIT, RES CMD, RES EXIT, RES MCMD, and RES VMCMD. A message is printed in the BBI-SS PAS Journal log informing you that the command would have been executed if this control request were not in effect. This is an easy way to test new EXECs.
NOLIST	Normal EXEC processing is in effect	Default
PERLIM	CPU percentage limit for the EXEC	If the CPU usage of an EXEC exceeds this value in any 15 second interval after the EXEC begins, the EXEC will be terminated. The CPU percentage is calculated based on the total CPU time available on 1 CPU within that 15 second interval. For example, 20% means 20% of 15 seconds. If the CPU time exceeds 3 seconds with any given 15 second interval, the EXEC will be terminated. The maximum CPU percentage usable is 100%, even on multiprocessor machines. Specifying 0 means no CPU percentage checking is performed.

Parameter	Function	Notes
TIMLIM	CPU time limit for the EXEC	If the EXEC exceeds this value in CPU seconds, it is terminated. Specifying 0 means no CPU time checking is performed.
SELLIM	Limits the number of nested EXECs in the current EXEC thread	This applies only to nested EXECs invoked with the IMFEXEC SELECT command using WAIT(YES). Specifying 0 means no limit checking is performed.
MAXTPUT	Limits the number of TPUTs that can be issued from the current EXEC	TPUTs occur when a REXX EXEC uses the TRACE command and when there are TSO/E error messages. Specifying 0 means no limit checking is performed.
Note: The use of the PERLIM, TIMLIM, SELLIM, and MAXTPUT parameters in an EXEC will temporarily override corresponding parameters set in the BBPARM member (as well as those dynamically updated using the Dynamic Parameter Manager application) as follows: PERLIM PEREXLIM TIMLIM TIMEXLIM SELLIM SELLIM MAXTPUT MAXTPUT		
GLOBAL	Propagate CNTL settings to all called EXECs	When set, any EXEC invoked through the IMFEXEC SELECT command inherits the CNTL settings of the current EXEC.
LOCAL	All settings are local to this EXEC	CNTL settings will not be propagated to EXECs called by this EXEC.

Condition codes are listed in the following table.

Value	Description
0	Command was executed
8	Invalid syntax was used

Example

This command causes echoing of all IMFEXEC commands and the termination of the EXEC if its total CPU time exceeds 5 seconds.

```
/* REXX */
"IMFEXEC CNTL LIST TIMLIM(5)"
```

DOM

This command deletes a WTO or WTOR.

Command	Parameters
DOM	ID(domid)

Certain descriptor codes indicate to MVS that a WTO should not roll off the master console but should stay there until explicitly deleted by the operator. If the cause for this WTO is no longer present, or the reply for a WTOR is no longer required, the WTO(R) can be deleted using the IMFEXEC DOM command.

Two AutoOPERATOR variables are used to determine the domid of a WTO or WTOR, or Rule-initiated EXECs. IMFWTDOM is set when an IMFEXEC WTO or IMFEXEC WTOR command is issued. IMFDOMID is set for Rule-initiated EXECs triggered by WTOs, WTORs, or MWTO events. It may be necessary to save IMFWTDOM or IMFDOMID in a shared variable so that a subsequent EXEC can issue the DOM.

The following table describes the parameters.

Parameter	Function	Notes
ID(domid)	The WTO(R) Sequence number	It uniquely identifies a WTO(R) and can be retrieved either: • In Rule-initiated EXECs (that are triggered by the event types WTO, WTOR, or MWTO) using the IMFDOMID variable • In an EXEC that issues IMFEXEC WTOs or WTORs from IMFWTDOM.

Condition codes are listed in the following table.

Value	Description
0	DOM issued
8	DOM ID missing

Example

This example deletes the WTO(R) that caused the EXEC to be triggered.

```
/* REXX */
"IMFEXEC DOM ID("IMFDOMID")"
```

DOMID for a WTO or WTOR that is issued by an EXEC is placed in variable IMFWTDOM.

EXIT

This command sets the return code IMFRC.

Command	Parameters
EXIT	[CODE(0|n)]

The following table describes the parameters.

Parameter	Function	Notes
CODE	The return code to be passed back to the invoker	This return code will be passed on to an IMFSUBEX program or an EXEC in the variable IMFRC.

Note: This command does not terminate the EXEC. You must explicitly terminate the EXEC using standard REXX constructs.

Condition codes are listed in the following table.

Value	Description
0	This command always returns a zero condition code

Example

This example command terminates the current EXEC and signals a return code (IMFRC) of 12.

```
/* REXX */
"IMFEXEC EXIT CODE(12)"
```

HB

This command changes the interval between heartbeat messages exchanged by a BBI-SS PAS and the Elan workstation.

Command	Parameters
HB	[INTERVAL(30|n)]

The following table describes the parameters.

Parameter	Function	Notes
INTERVAL	The number of seconds between heartbeat messages exchanged between the BBI-SS PAS and the Elan Workstation	This command changes the interval for both the BBI-SS PAS and the Elan Workstation. The change takes effect after the expiration of the current interval.

Condition codes are listed in the following table.

Value	Description
0	Command successfully executed
8	Interval specification missing or invalid

Example

This command forces heartbeat messages between the BBI-SS PAS and the Elan Workstation to be exchanged every 20 seconds.

```
/* REXX */
"IMFEXEC HB 20"
```

IMFC

This command issues an IMF or MAINVIEW for DB2 service command to:

- Invoke an analyzer display
- Start and stop monitors
- Invoke monitor displays (such as PLOT and DMON)
- Invoke a display for automatic image logging

This command supports only local targets. Local targets are assigned in BBPARM member BBIJNT00 (refer to the *MAINVIEW Common Customization Guide* for more information about this BBPARM member).

Command	Parameters
IMFC	Command/options TARGET|IMSNAME=, [IMAGE=,] [USRID=,] [SCROLL=YES|NO]

The following table describes the parameters.

Parameter	Function	Notes
Command/options	An IMF or MAINVIEW for DB2 service command and any required parameters	1- to 8 alphanumeric characters.
TARGET= IMSNAME=	The target for the request This command supports only local targets. Local targets are assigned in BBPARM member BBIJNT00 (refer to the *MAINVIEW Common Customization Guide* for more information about this BBPARM member).	An IMSID (or alias) value can be used as the TARGET name if it has been specified in the BBIJNT00 member of BBPARM. If TARGET= is not specified, a PM0330E error message is logged in the BBI-SS PAS Journal log and the request is terminated.
IMAGE=	Write output to the Image Log (BBIIMAGx)	YES or NO. YES is the default.
USRID=	The user ID to be associated with the command.	The user ID is checked when requests to purge monitor services are processed. The default is AUTOID from BBIISP00 or the characters USRID.
SCROLL=	Scrolls the display forward one full page (40 lines) from the previous IMFC request if scrolling is available in that particular display	YES or NO. NO is the default.

If you use the parameters `SCROLL=YES,IMAGE=NO`, the data retrieved will contain every screen one by one until the line `END OF DATA` is found. If you use the parameters `SCROLL=YES,IMAGE=YES`, all screens are logged but the data contains only the last screen.

Examples

This section contains examples using the IMFEXEC IMFC command statement. A brief discussion follows each example.

Note: Remember that the IMFC statement does not return a value in IMFNOL.

Example 1

```
/* REXX */

LAST = ''    /* FOR THE FIRST TIME */

LOOP:

IF LAST = '' THEN "IMFEXEC IMFC OSTAT LTERM=B* TARGET=IMS41X"
ELSE "IMFEXEC IMFC OSTAT LTERM=B* START="LAST" TARGET=IMS41X"

I = 4
DO WHILE I <= 24
 "IMFEXEC VGET LINE"I" LOCAL"
 IF SUBSTR(VALUE('LINE'I),67,9) = 'CONNECTED' THEN
 DO
 TERM = SUBSTR(VALUE('LINE'I),2,8)
 IF 'LINE'I = 'LINE24' THEN NOP    /* LINE24 WILL BE INCLUDED IN NEXT */
 ELSE
 "IMFEXEC MSG 'TERMINAL "TERM" IS NOT CONNECTED'"
 END
ELSE NOP
I = I + 1
END
LAST = SUBSTR(VALUE('LINE24'),2,8)  /* LAST TERMINAL ON THE SCREEN */

IF LAST ,= '' THEN SIGNAL LOOP;
ELSE EXIT
```

This example shows a REXX EXEC issuing the IMFC OSTAT service. The EXEC loops until there is no more data and checks the terminals with the status: NOT CONNECTED. For these terminals, it issues message on the BBI LOG.

Example 2

```
/* REXX */
"IMFEXEC VGET QIMSNAME"
"IMFEXEC IMFC USER RESPINP TARGET="QIMSNAME" IMAGE=NO"
DO I = 8 TO 43
 "IMFEXEC VDCL IMFL"I" LIST(V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11)"
 "IMFEXEC VGET LINE"I" INTO(IMFL"I") LOCAL"
 "IMFEXEC MSG .. "V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11"'"
 IF SUBSTR(V1,1,3) = '***' THEN EXIT
END
DO 900
   "IMFEXEC IMFC USER RESPINP TARGET="QIMSNAME" IMAGE=NO SCROLL=YES"
   DO I = 8 TO 43
   "IMFEXEC VDCL IMFL"I" LIST(V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11)"
   "IMFEXEC VGET LINE"I" INTO(IMFL"I") LOCAL"
   "IMFEXEC MSG .. "V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11"'"
   IF SUBSTR(V1,1,3) = '***' THEN EXIT
   END
END
```

This example shows a REXX EXEC that issues the IMFC USER service with a parameter of RESPINP. The RESPINP parameter is used for IMS terminals in response input mode. The USER service is a scrollable service.

The REXX EXEC invoked with the SELECT command, FREERSP, issues IMS commands to display, stop, dequeue and start the terminals. Note that for multiple screens of data returned, the IMFC service returns the screen in local variables LINE8 through LINE43, then places the next screen in the same variables, LINE8 through LINE43.

IMFC SET PRG=CALLX|ALL

This command uses SET PRG=CALLX|ALL to terminate a time-initiated EXEC.

Command	Parameters
IMFC SET PRG=CALLX| ALL	Execname [USRID=userid,] TARGET=target name

The following table describes the parameters.

Parameter	Function	Notes
USRID=	The name of the user ID against which authorization checking is performed (if this is different from the user ID of the originator of the request).	If the user ID is different from the user ID of the request's originator, the purging user ID must have authority to purge the originator's CALLX requests. To accomplish this, code PMACC=# on the AUTHJOB= statement in the BBPARM authorization member for the appropriate user ID.
Note: If PRG=ALL is specified, security is done against the user ID that is specified on the AUTOID parameter for the BBPARM member BBIISP00.		
TARGET=	The target against which the CALLX request will be purged.	

Condition codes are listed in the following table.

Value	Description
0	Command executed successfully
8	One of the following: • TARGET= is missing • An error in the monitor or analyzer service occurred (Msg PM0334E issued) • Syntax error in SET command (Msg PM0337E issued)
16	Handling program not found

Example

This example shows how to use the IMFEXEC IMFC SET PRG=CALLX command statement.

```
"IMFEXEC IMFC SET PRG=CALLX IMSCHECK"
```

This example terminates a time-initiated EXEC called IMSCHECK. It can be called from a Rule when IMS terminates.

IMFC SET REQ=CALLX

This command uses SET REQ=CALLX to start a time-initiated EXEC or SET PRG=CALLX to terminate a time-initiated EXEC.

Command	Parameters
IMFC SET REQ=CALLX	Execname [START=hh:mm::ss,] [STOP=hh:mm:ss,] [STOPCNT=( )] [I=00:01:00|hh:mm:ss,] [USRID=userid,] TARGET=target name

The following table describes the parameters.

Parameter	Function	Notes
Execname	The name of the EXEC to be scheduled	No parameters can be passed to the EXEC.
START=	The start time for an EXEC	Format is: HH:MM:SS.
STOP=	The stop time for rescheduling the EXEC	Format is: HH:MM:SS.
STOPCNT=	The number of times the EXEC will be scheduled	A valid decimal number.
I=	The interval between EXEC schedules	Format is HH:MM:SS.
TARGET=	The target against which the EXEC will be scheduled	
USRID=	The name of the user ID for the request using the online application	The value specified will be used for authorization checking. It also will become the owner of the resulting request.

CALLX requests can be viewed and purged when you select the TIMEXEC option from the EXEC Manager Menu.

Condition codes are listed in the following table.

Value	Description
0	Command executed successfully
8	One of the following: • TARGET= is missing • An error in the monitor or analyzer service occurred (Msg PM0334E issued) • Syntax error in SET command (Msg PM0337E issued)
16	Handling program not found

Example

This example shows how to use the IMFEXEC IMFC SET REQ=CALLX command statement.

```
/* REXX */
"IMFEXEC IMFC SET REQ=CALLX @HOURLY START=06:00:00 STOP=16:00:00",
       "I=01:00:00  TARGET="IMFORGSS "USRID=JDB1"
```

This example causes an EXEC named @HOURLY to be issued every hour, beginning at 6:00 am and ending at 4:00 pm on the same system that this EXEC is invoked on. The user with user ID JDB1 will be able to purge this request. Also, any user with PMACC=# coded in the BBPARM user ID authorization member can purge this request.

IMSTRAN

This command submits IMS transactions.

Command	Parameters
IMSTRAN	Transaction code ['p1 ... pn']

The following table describes the parameters.

Parameter	Function	Notes
Transaction code and (optional) operands	Transaction code of transaction to invoke and optional parameters	Transaction code name must be defined in AAOTRN00. Any optional parameters must be included in quotes. Maximum length is 255 characters. If a command does not require an operand, you must code ' ' (a blank) as the operand. Conversational or remote transactions are not supported. In a shared queue environment, the combined length of all operands passed must not exceed 106 characters.

The response, if any, is queued to the LTERM specified in the RLTERM parameter of the BBPARM member AAOTRN00 of the target system. The default value is MASTER. You can set RLTERM=DFSMTCNT if you do not want any output to be sent back to the inputting LTERM.

Within IMS, the transaction is viewed as a normal transaction once it arrives in the message queue, with the input LTERM set to the RLTERM value. Only one destination LTERM is provided per BBI-SS PAS and IMS.

Condition codes are listed in the following table.

Value	Description
0	Command issued successfully.
12	An error occurred. The error message is logged to the BBI journal.

Example

This example invokes the IMS transaction ADDPART.

Note: If you need to continue this command, use the minus sign (-) as the TSO continuation character to avoid inserting a blank into the statement.

```
/* REXX */
"IMFEXEC IMSTRAN ADDPART 'AB960C10,RIVET,74'"
```

JES3CMD

This command issues a JES3 command through the subsystem interface. It should be used only for JES3 releases prior to 2.2. IMFEXEC CMD should be used for newer releases.

A response is not returned to the EXEC.

Command	Parameters
JES3CMD	'JES3 command'

The following table describes the parameters.

Parameter	Function	Notes
Command	The JES3 command to be issued	The maximum length is 127 bytes. The JES3 command character specified in BBPARM member BBISSP00 is automatically appended to the front of the command. Refer to the *MAINVIEW Common Customization Guide* for details.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
8	Invalid syntax used

Example

This example command displays JES3 status information about job PROD1234 on the JES console.

```
/* REXX */
"IMFEXEC JES3CMD I,J=PROD1234"
```

JESALLOC

Unlike the TSO ALLOCATE command, this command may be used to allocate a subsystem data set even when a JES connect was performed (such as, JES was started after the AutoOPERATOR subsystem).

Command	Parameters
JESALLOC	DDNAME [CLASS] [SYSOUT]

The following table describes the parameters.

Parameter	Function	Notes
DDNAME	Name of the DD statement that is be allocated.	The maximum length is eight characters and must conform to the DD name specifications.
CLASS	Class to be associated with this SYSOUT DD statement. Specify '*' for the MSGCLASS of the STC.	This parameter is a one-character valid JES output class.
SYSOUT	A literal indicating that this DD statement should be allocated to SYSOUT.	Code this parameter to be compatible with future extensions to this command.

Condition codes are listed in the following table.

Value	Description
0	Command successfully executed
4	JES rejected the allocation request
8	Not connected to JES (started under MSTR and no JESCNCT card in bbissp00)
16	Syntax error
20	DD name in use (use TSO free command first)

Example

This example shows how to use the IMFEXEC JESALLOC command statement.

```
/* REXX */
“IMFEXEC JESALLOC MYPRINT SYSOUT CLASS(A)”
```

Allocates a SYSOUT data set in the SYSOUT class A to the DD statement ‘MYPRINT’.

Note: You can use the TSO FREE command to free this DD statement again.

JESSUBM

Unlike the TSO SUBMIT command, this command may be used to allocate a subsystem data set even when a JES connect was performed (such as, JES was started after the AutoOPERATOR subsystem). Two advantages of this command are as follows:

- It can submit a job directly from variables.
- It returns the job number of the submitted job in a variable.

Note: The JESSUBM command will not drive the TSO SUBMIT exit IKJEFF10.

Command	Parameters
JESSUBM	DSNAME | DS | DA | DSN STEM

The following table describes the parameters.

Parameter	Function	Notes
DSNAME | DS | DA | DSN	Name of the data set to submit. Sequential and partitioned data sets are supported. When specifying a partitioned data set, a member name must be supplied.	Length can be from 1 to 44 characters conforming to data set name specifications. The data set must have an LRECL of 80.
STEM	Set of REXX stem variables that contain the JCL to be submitted.	Length can be from 1 to 26 characters conforming to variable naming conventions. The separator character between variable name and index is '.' (according to REXX stem variable syntax). The variable with the index 0 is assumed to contain the count of variables to be processed. Note that the variable contents should not exceed 80 characters in length. Contents in columns 72 through 80 is accepted.

Note: STEM() and DSN() are mutually exclusive; however, one or the other must be specified.

Condition codes are listed in the following table.

Value	Description
0	Command successfully executed
4	JES rejected the allocation request
8	INTRDR cannot be dynamically allocated. This error can happen if JES has not yet started.

Value	Description
12	Specified data set cannot be allocated or opened or data set LRECL is less than or greater than 80, or data set name is too long.
16	Syntax error occurred.
20	Error processing input variables.
24	Not connected to JES (started under MSTR and no JESCNCT card is in BBISSP00).
28	Invalid input data set or error writing to INTRDR.

After a successful submit, the variable IMFJESNR is set to the job ID of the submitted job (for example, JOB12345). If multiple jobs were submitted as a stream, this variable contains the job ID of the first job in this stream.

Note: Use of this command is recommended over the traditional IMFEXEC SUBMIT command.

Example

Examples of the IMFEXEC JESSUBM command statement follow.

Example 1:

```
/* REXX */
“IMFEXEC SUBMIT DA(’BAORAE.JCL.CNTL(IEFBR14)’)”
```

Submits the JCL contained in the PDS member ‘BAORAE.JCL.CNTL(IEFBR14)’

Example 2:

```
/* REXX */
S.1='//BAOBR14  JOB (3911),'ERNST',CLASS=K,MSGCLASS=A,NOTIFY=BAORAE2'
S.2='//IEFBR14  EXEC PGM=IEFBR14'
S.0=2
"IMFEXEC JESSUBM STEM(S)"
```

Submits the JCL contained in the variables S.1 and S.2.

LOGOFF

This command terminates the connection between an EXEC and an OSPI session. If the DISCONNECT parameter is not specified, it also logs off the application and frees all internal resources associated with the session.

Refer to “Interacting with VTAM-Applications with OSPI” on page 113 for more information about using this command and OSPI.

Command	Parameters
LOGOFF	SESSION() [DISCONNECT]

The following table describes the parameters.

Parameter	Function	Notes
SESSION()	Specifies the session identifier that is returned when establishing a session with the LOGON command	Determined by the results of the LOGON command.
DISCONNECT	Request temporary disconnection of the session	If specified, retains the session in the background so it can be picked up by a later LOGON command. Otherwise, it issues a TERMSESS macro against the application and closes the VTAM ACB that communicates with the application. This results in an unconditional LOGOFF from the application. All internal resources associated with this session are freed.

Condition codes are listed in the following table.

Value	Description
0	Command executed successfully
8	Syntax error or indicated session not found

Example

This example command terminates a session previously established using a LOGON command with a PREFIX() parameter specifying TSO.

```
/* REXX */
"IMFEXEC LOGOFF SESSION("TSOSESS")"
```

LOGON

This command establishes a session between an EXEC and a VTAM application and supplies the first screen output to the EXEC.

Refer to “Interacting with VTAM-Applications with OSPI” on page 113 for more information about using this command and OSPI.

Command	Parameters
LOGON	[APPLID|ACB(Application name)] [DATA|USERDATA(Userdata)] [PREFIX(OSI|Prefix)] [SESSION(Session identifier)] [REQACB(ACB to use)] [LOGMODE(D6327802|Logmode)] [DEBUG|NODEBUG] [NORECEIVE]

The following table describes the parameters.

Parameter	Function	Notes
APPLID|ACB	The ACB name of the application (to establish a session with) as it is specified in SYS1.VTAMLIST	Required if the SESSION parameter is not specified. 1-8 alphanumeric characters. The application must be active and accepting LOGONs.
DATA| USERDATA	Any data passed to the application during logon processing	Maximum length is 80 characters.
PREFIX	OSPI variable name prefix	Must be exactly 3 characters long. The first character must be an alpha character.
SESSION	Session identifier of a previously disconnected session	When specified, this parameter indicates that no new session should be established but an existing session reconnected.
REQACB	ACB to use to communicate with the application	Since the ACB name used will also represent the terminal name when connecting to an application, this parameter may be used for applications which allow access only from specific terminals. If this ACB is unavailable for whatever reason, the command will fail.
LOGMODE	Logmode to use when requesting a session	The logmode determines the screen characteristics to emulate (in particular, the virtual screen size).

Parameter	Function	Notes
DEBUG|NODE- BUG	Controls whether or not execution of all activities against the resulting session will execute in DEBUG mode	When specified, many messages about internal activities are generated and buffer snaps are taken.
NORECEIVE	Specify this parameter when logging on to an application that does not display an initial panel before allowing the terminal user to enter data.	

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
4	A LOGON with the SESSION parameter was issued but failed to reestablish the session
8	Syntax error or LOGON failed

Example

This example command establishes a session to the application BTSO and passes the string SYSUSER to it. All control variables have the prefix TSO and no debugging messages are generated.

```
/* REXX */
"IMFEXEC LOGON APPLID(BTSO) DATA(SYSUSER) LOGMODE(D6327803) PREFIX(TSO)"
"IMFEXEC VGET TSOSESS LOCAL"
```

Following the successful execution of this LOGON command, the variable xxxSESS (where xxx represents the session prefix) must be retrieved from the local variable pool. If it is not retrieved, you cannot perform further commands against this session.

The token contained in this variable uniquely identifies the session. The default name for this variable is OSISESS. Refer to “Interacting with VTAM-Applications with OSPI” on page 113 for more information.

MSG

This command logs a message in the BBI-SS PAS Journal log.

Command	Parameters
MSG	'Message text' [TARGET(Target name)]

The following table describes the parameters.

Parameter	Function	Notes
Message text	Text of the message to issue	Maximum length is 252 bytes.
TARGET	Target system name	1-8 alphanumeric characters. Valid target names are defined in the BBIJNT00 member of the BBPARM data set.

Note: Specifying a null variable for Message text causes an error.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
8	NODE is not found when TARGET is used (check BBINOD00 in BBPARM)
12	TARGET is not found when target is used (check BBIJNT00 in BBPARM)
16	NODE is not available

Example

This example sends a message to the BBI-SS PAS monitoring the target named CICSPRDA. The message is logged on the remote Journal and no entry is made on the originating system's Journal.

```
/* REXX */
"IMFEXEC MSG 'MANUFACTURING DATABASE IS OFFLINE' TARGET(CICSPRDA)"
```

NOTIFY

This command sends a request through AutoOPERATOR to issue a pager call to the AutoOPERATOR Elan Workstation.

Command	Parameters
NOTIFY	NAME(Phone number) [INFO('Text')]

The following table describes the parameters.

Parameter	Function	Notes
NAME	The contact name defined to the Elan Workstation	1-32 characters alphanumeric. Elan equates this name to a telephone number to be dialed.
INFO	Any information to be passed and placed on the pager	1-12 alphanumeric characters. Text must be included in quote marks.

Condition codes are listed in the following table.

Value	Description
0	Elan successfully passed the information
8	The request timed out
12	Elan could not execute the request
16	Elan communications were not established

Example

This example command notifies the individual SYSPROG through the Elan Workstation, passing the information SYSTEM to the pager.

```
/* REXX */
"IMFEXEC NOTIFY NAME(SYSPROG) INFO(SYSTEM)"
```

POST

This command notifies an EXEC that has issued the IMFEXEC WAIT command that it can resume execution (for example, makes that EXEC dispatchable again).

Command	Parameters
POST	name CODE(code) TARGET(targetname)

The following table describes the parameters.

Parameter	Function	Notes
name	Specify a name that matches the name specified by another EXEC using the IMFEXEC WAIT command The value of this parameter must match the NAME parameter of a previously executed IMFEXEC WAIT command. Refer to “WAIT” on page 404 for more information.	1-32 alphanumeric characters.
CODE(code)	Optional. Can be 1-255 characters If the code contains blanks, it must be entered in single quotation marks. Available to the reawakened EXEC in the TSO variable IMFPOST.	1-255 alphanumeric characters.
TARGET(target)	Target system name Valid target names are defined in the BBIJNT00 member of the BBPARM data set.	1-8 alphanumeric characters.

Condition codes are listed in the following table.

Value	Description
0	Name successfully posted
4	No waiting EXEC found
8	Node not found
12	Target not found
16	Node not available

Example

This example posts the name TEST with a code of ABC to the target called SYSA. An EXEC that is waiting for TEST to be posted will now be reenabled when TEST is posted. The value of the code can be examined by the reenabled EXEC by using the variable IMFPOST.

```
"IMFEXEC POST TEST CODE(ABC) TARGET(SYSA)"
```

RECEIVE

This command issues a VTAM RECEIVE against an OSPI session. It is used for OSPI sessions with applications that use non-standard protocol.

Refer to “Interacting with VTAM-Applications with OSPI” on page 113 for more information about using this command and OSPI.

When using this statement, you must remember to code IMFEXEC with the RECEIVE command. If you do not, you might cause the TSO/E RECEIVE command to be invoked.

Command	Parameters
RECEIVE	[TIMEOUT(10|n)]

The following table describes the parameters.

Parameter	Function	Notes
TIMEOUT	The time to wait, in seconds, for data to arrive	Numeric value in the range 0-9999.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully, data was received
4	No data was available during the given interval

Example

This command attempts to receive another data packet from the session identified by the CNMSESS variable.

```
/* REXX */
"IMFEXEC RECEIVE SESSION("CNMSESS")"
```

RES

This command executes basic SYSPROG services commands.

To use this command, you must have the AutoOPERATOR for MVS component installed. If SYSPROG is installed but AutoOPERATOR for MVS is not, this command will not function. When AutoOPERATOR for MVS is not installed, use the IMFEXEC CMD (MVS command with response) command to access basic SYSPROG services as an MVS started task using an MVS MODIFY command to the SYSPROG services task.

Command	Parameters
RES	SYSPROG command or 'SYSPROG command' WAIT(60|nnnn)

Output from SYSPROG service commands is placed in LOCAL variables LINE1 thru LINEnn, where IMFNOL contains the number of lines returned. Parentheses in the output are removed. These variables must be retrieved using an IMFEXEC VGET command before they can be used in an EXEC.

The following table describes the parameters.

Parameter	Function	Notes
SYSPROG service command	The SYSPROG service command, including any required parameters	Any supported SYSPROG services command.
WAIT(60|nnnn)	Amount of time to wait for a response from the command	The default is 60 seconds or nnnn, which can be a value from 0 to 9999 seconds.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
8	No service parameter was passed when required
12	Command timed out in interface to SYSPROG service (the response time exceeded 60 seconds). **Note:** SYSPROG service commands that require a user response (for example, CHAP) cause a timeout in the SYSPROG service interface. Refer to the *RESOLVE PLUS Reference Manual* for more information about how to determine if a command requires a response.
20	AutoOPERATOR for MVS is not installed.

Example

The output from this example is automatically put into variables named LINE1 - LINEnn, where nn is the number of lines in the output. The variable ASML1 is declared to contain a list of seven variables. The first line of the output is placed into ASML1, which parses the line automatically into the seven variables.

```
/* REXX */
"IMFEXEC RES ASM"
"IMFEXEC VDCL ASML1 LIST(V1 V2 V3 V4 IPLTYPE V6 V7)"
"IMFEXEC VGET LINE1 INTO(ASML1) LOCAL"
"IMFEXEC VPUT IPLTYPE"
"IMFEXEC MSG 'AN IPL "IPLTYPE "START WAS PERFORMED’”
```

The IPLTYPE variable is put into the SHARED variable pool and a message is issued to the Journal to indicate that a certain type of IPL start was performed.

SCAN

This command performs any of the following actions on the VTAM buffer image of an OSPI session:

- Finds a specific character string
- Positions the cursor in the first input field following a specified character string
- Retrieves data from the buffer image into user-defined variables To perform all three functions, you must use the parameters SESSION, ROW, and COLUMN. The default for ROW and COLUMN is 1. There is no default for SESSION.

For more information about performing these functions and which additional parameters you must use, refer to “Using Parameters” on page 350.

Refer to “Interacting with VTAM-Applications with OSPI” on page 113 for more information about using this command and OSPI.

Command	Parameters
SCAN	[ROW(Starting row)] [COL(Starting column)] [TEXT(Target text)] **VARI**ABLE(Variable name)|[POSITION] [LENGTH(n)] SESSION(Session identifier) [CASE|NOCASE] [TRIM|NOTRIM]

The following table describes the parameters.

Parameter	Function	Notes
ROW	The row in which to begin the scan	Numeric value in the range one through the maximum number of rows supported by the emulated terminal. Default is 1.
COL	The column in which to begin the scan	Numeric value in the range 1-80. Default is 1.
TEXT	The text to scan for	Maximum length is 255 characters **Note:** TEXT and VARIABLE cannot be coded together on the same statement.
VARIABLE	The name of the variable to receive the data	Do not specify a leading ampersand (&). Either VAR or POSITION or both must be specified. **Note:** TEXT and VARIABLE cannot be coded together on the same statement.
LENGTH	The number of characters to place into the target variable	Required with VAR. Numeric value in the range 1-255.

Parameter	Function	Notes
SESSION	Session identifier of the OSPI session that should be accessed	Provided by the LOGON command.
CASE	Performs a case-sensitive scan for TEXT	Default is NOCASE.
NOCASE	Performs a non-case-sensitive scan for TEXT	
TRIM	All leading and trailing blanks, nulls, and control characters are to be removed from the returned variable	Default is NOTRIM
NOTRIM	The data is placed into the target variable exactly as found	
POSITION	The cursor is automatically positioned in the next input field following the indicated character string	Either VAR or POSITION or both must be specified.

Using Parameters

This section describes which parameter you must use to perform certain functions with IMFEXEC SCAN.

Searching for string and positioning the cursor: The following additional parameters must be used with IMFEXEC SCAN when you want to either search for a string or position the cursor:

- TEXT

  Indicates a request to find a text string.

  The parameters VARIABLE, LENGTH, and TRIM do not apply when searching for text (and can, in some cases, cause syntax errors). Do not specify any of these parameters when you use TEXT.

- CASE

  Indicates whether to perform a case-sensitive search for TEXT keyword.

- POSITION

  Indicates whether or not to position the cursor. If POSITION is not specified, the position will not changed.

Retrieving data into variables: The following additional parameters must be used with IMFEXEC SCAN when you want to retrieve data into variables:

- VARIABLE

  Indicates a request to retrieve data.

  The parameters TEXT, CASE, and POSITION do not apply when retrieving data (and can, in some cases, cause syntax errors). Do not specify any of these parameters when you use VARIABLE.

- LENGTH

  This is required to retrieve data.

- TRIM

  Indicates whether or not to remove trailing blanks, nulls, and control characters from the returned variable.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully; text was found
4	Text not found
8	One of the following: • Syntax error • Conflicting parameters specified • Session not found

Examples

This section contains examples using the IMFEXEC SCAN command statement. A brief discussion follows each example.

Example 1

```
/* REXX */
"IMFEXEC SCAN TEXT(USERID) VAR(MYUSER) LENGTH(25) TRIM NOCASE",
 "SESSION("TSOSESS")"
```

This example command scans the screen beginning at the upper left-hand corner for the character string, USERID. The scan is performed against the session designated by the TSOSESS variable.

The scan is not case-sensitive. The 25 characters following the string are placed into the variable MYUSER after trailing and leading blanks have been removed.

Example 2

```
/* REXX */
"IMFEXEC SCAN ROW(10) COL(10) POSITION TEXT(A) LENGTH(1)",
 "SESSION("OSISESS")"
```

This example places the cursor in the input field after row 10, column 10, for the virtual screen buffer with the logical unit specified by the variable OSISESS. This command is equivalent to placing the cursor on row 10, column 10, and pressing the TAB key.

SELECT

This command invokes an EXEC or a program.

Command	Parameters
SELECT	EXEC(Execname [p1 ... pn])|PGM(Program name) [PARM('p1 ... pn')] [PRI(NORMAL|HIGH)] [WAIT(NO|YES)] [TARGET(Target system)]

The following table describes the parameters.

Parameter	Function	Notes
EXECname	Name of EXEC to invoke, including all parameters	Maximum length is 255 characters. Either EXEC or PGM must be specified.
PGM	The name of a user-written routine stored in the BBLINK data set on the local BBI-SS PAS. Refer to “Using Other Programming Languages” on page 354 for more information.	The name of the routine must begin with the prefix IMFUxxxx. Either PGM or EXEC must be specified.
PARM	A list of parameters to be passed to the program	
PRI	Execution priority of the EXEC to be invoked	Either NORMAL or HIGH. Applies only to EXEC keyword. It overrides AAOEXP00 parameters. PRI is valid with WAIT(NO) but not with WAIT(YES). PRI is ignored with WAIT(YES).

Parameter	Function	Notes
WAIT	Suspension criterion for invoking EXEC	Either YES or NO. NO causes the EXEC to be queued for execution using a different EXEC thread. YES causes the EXEC to execute under the same thread as the calling EXEC. If YES is specified, the invoking EXEC waits for the invoked EXEC to terminate. WAIT(YES) is required to retrieve a return code from the invoked EXEC. WAIT(YES) is not supported for EXECs scheduled to a remote system with TARGET. If both WAIT and TARGET are specified, the EXEC is queued to the TARGET
TARGET	Identifies the target SS on which the command will be executed	The name must be defined in the BBPARM member BBIJNT00 as either the target name or BBI-SS PAS subsystem ID.

Note: If you use the IMFEXEC SELECT command to schedule an EXEC and you do not specify a target and WAIT(YES) is specified, the LOCAL variable pool is shared between the calling EXEC and the called EXEC.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
8	One of the following is true: • Target specified is not found in BBIJNT00 • EXEC specified but is not found in BBPROC • Program name length is greater than 8 characters • Program is not found
12	One of the following is true: • EXEC name specified is more than 8 characters long • Program does not start with IMFU
16	Invalid syntax used

Example

This example command invokes the EXEC CHKENQ on the remote SS SYSB, passing it the parameter SYS2.PROD.XLIB.

```
/* REXX */
"IMFEXEC SELECT EXEC(CHKENQ SYS2.PROD.XLIB) TARGET(SYSB)"
```

This section also contains a discussion about how you can use other programming languages when you use the IMFEXEC SELECT command and how to determine condition codes when you select other program to run from an EXEC. Refer to “Using Other Programming Languages” on page 354 and “Understanding Completion Codes for EXEC-Initiated EXECs with WAIT(YES) and User Written Programs” on page 355.

Using Other Programming Languages

Programming languages in addition to REXX and CLIST, such as Assembler, PL/I and COBOL, may be used to implement complex automation tasks. These programs are called user-written programs. A user-written program is called only from an EXEC, is loaded from the BBILOAD library in the BBI-SS PAS, and must begin with the prefix IMFUxxxx to prevent a conflict with future AutoOPERATOR program names.

At entry the program will be given a parameter list specifying the name of the EXEC, the originating BBI-SS PAS Address Space ID, and the contents of the parameter string.

Parameter	Description
WORD1	A count of the number of parameters (to maintain compatibility with PL/I). This number is always 5.
WORD2	A pointer to an 8-character field containing the name of the EXEC scheduling the program, left-justified.
WORD3	A pointer to a 4-character field containing the BBI-SS PAS Address Space ID of this BBI-SS PAS, left-justified.
WORD4	The length of the parameter string pointed to by WORD5.
WORD5	A pointer to the parameter string.

This program gains control in KEY 8, problem state, and is afforded ESTAE protection by AutoOPERATOR. The execution of the calling EXEC is suspended until the User-written program terminates. The program inherits the APF authorization of the subsystem.

The high-order bit of the last word in the parameter list is set to 1. The program should be coded and link edited as re-entrant because it could be called from several tasks. If serialization is required, it must be provided by the user program using ENQ facilities of MVS. The program can be coded to execute in either 24-bit or 31-bit mode.

A user-written program may access and manipulate TSO variables. BMC Software recommends that you be familiar with TSO internals before attempting this. One documented programming interface for manipulating TSO variables is the IKJCT441 program.

The following describes the register contents on entry to a user-written program:

Register Entry	Description
R1	Pointer to the parameter list
R2 - R12	Unpredictable
R13	An 18-word save area
R14	The return address to AutoOPERATOR
R15	The entry-point address of the user exit

The following describes the register contents expected by AutoOPERATOR when control is returned from the user-written program.

Register Exit	Description
R15	The return code made available to the calling EXEC in the BBI variable &IMFRC

AutoOPERATOR expects the program to return control in problem state, KEY 8. If the program abends, IMFCC is set to 20.

Understanding Completion Codes for EXEC-Initiated EXECs with WAIT(YES) and User Written Programs

Both EXEC-initiated EXECs with WAIT(YES) and user-written programs gain control of the thread while the execution of the initiating EXEC is suspended. When the execution of the initiating EXEC is resumed the initiating EXEC can determine the success of the called EXEC or program by testing the BBI variables IMFCC and IMFRC.

If IMFCC is zero, meaning the EXEC (with the WAIT(YES) parameter) or user-written program was successfully invoked, a separate local variable, IMFRC, will be set with the return code from the program or EXEC. A selected EXEC with WAIT(YES) can return this value by using the EXIT command. See “EXIT” on page 324. For example:

```
IMFEXEC EXIT CODE(12)
```

causes IMFRC to be set to 12 when the calling EXEC receives control.

The following values of IMFCC are valid for SELECTed EXECs and programs:

Value	Description
00	Program or EXEC scheduled
08	Program or EXEC not found
20	Severe error

SEND

This command sends a message to a TSO or IMS user.

Command	Parameters
SEND	'Msgtext' LTERM(Terminal)|USER(TSO USERID)

The following table describes the parameters.

Parameter	Function	Notes
Msgtext	The message to be sent to the user	The maximum length of the message text is 252 bytes when LTERM is coded; otherwise, the maximum length is 120 bytes. This includes the SEND ' ' USER( ) portion of the command.
LTERM	The IMS Lterm to receive the message	Either LTERM or USER must be specified.
USER	The TSO user ID to receive the message	Either LTERM or USER must be specified.

Note: The parameters of the TSO SEND command LOGON, SAVE, WAIT, and so on are not supported. To perform similar functions, use the AutoOPERATOR IMFEXEC command statement IMFEXEC CMD #SE.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
8	Length of command exceeds maximum
12	No destination was supplied
16	Invalid syntax used

Examples

This section contains examples using the IMFEXEC SEND command statement. A brief discussion follows the examples.

Example 1

```
/* REXX */
"IMFEXEC SEND 'I AM SENDING YOU THIS MESSAGE' USER(CWB1)"
```

This command sends a message to the TSO user CWB1. If the user is not currently logged on, the message will be discarded.

Example 2

```
/* REXX */
"IMFEXEC SEND 'I AM SENDING YOU THIS MESSAGE' LTERM(R35769D)"
```

This command sends a message to the IMS LTERM R35769D of the IMS system the BBI-SS PAS is connected to.

SESSINF

This command writes OSPI session specific information to the OSPISNAP DD and is used in debugging OSPI EXECs. Information includes the current contents of the buffer image, cursor position and keyboard status, and a SNAP dump of all VTAM data exchange, including the RPL.

Refer to “Interacting with VTAM-Applications with OSPI” on page 113 for more information about using this command and OSPI.

Command	Parameters
SESSINF	SESSION(Session identifier)

The following table describes the parameters.

Parameter	Function	Notes
SESSION	The session identifier of the session to display	This identifier is returned by the LOGON command. Only one EXEC can use this command at a time. You must serialize the use of this command.

Note: The SNAP dump information produced is for BMC Software support purposes only.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
8	Indicates one of the following: • Invalid syntax was used • Session not found

Example

This example command dumps all session related information designated by IMSSESS to the OSPISNAP DD. This includes all session variables and the virtual screen image.

```
/* REXX */
"IMFEXEC VENQ 'BOOLE' EXC"
"IMFEXEC SESSINF SESSION("IMSSESS")"
"IMFEXEC VDEQ 'BOOLE'"
```

SETTGT

This command resets the EXECs target and adjusts the contents of the IMFSYSID variable.

Command	Parameters
SETTGT	'Target name'

The following table describes the parameters.

Parameter	Function	Notes
Target name	The system ID of the new target	1-8 alphanumeric characters. Only targets assigned to the BBI-SS PAS that this EXEC is currently running on are acceptable. Valid targets and their BBI-SS PAS assignments are specified in the BBIJNT00 member of the BBPARM data set. Value must be enclosed in single quotes.

Example

This example command shows how CICS task information is retrieved from the CICSPROD system and CICS terminal information is retrieved from the CICSTEST system.

```
/* REXX */
"IMFEXEC SETTGT 'CICSPROD'"
"IMFEXEC CICS QUERY TASK"
"IMFEXEC SETTGT 'CICSTEST'"
"IMFEXEC CICS QUERY TERMINAL"
```

Important Note
The IMFEXEC SETTGT command statement is used for CICS regions **only**. To set a target which is not CICS region, use: `"IMFEXEC SELECT EXEC(execabc) TARGET(tgtname)"`

SHARE

Scans the LOCAL pool for matching variable names. Every match is recorded. At the end of the EXEC the values for all these matches are gathered and transferred back to the calling EXEC (AOEXEC command). The use of this command is only meaningful in an EXEC that is driven by AOAnywhere.

Command	Parameters
SHARE	Variable name|(var1 var2...var*n*)

The following table describes the parameters.

Parameter	Function	Notes
Variable	Name of a variable or variables that will be searched for in the LOCAL variable pool and transferred back to the calling EXEC (AOEXEC command).	You can list one variable or a list of variable names. Multiple variable names must be enclosed in parentheses. You can list either fully qualified variable names or variable name patterns. Patterns must follow coding conventions such as • A*B+ • A* • * This command overrides any VAR() parameters specified on the AOEXEC SELECT statement.

At the time of the IMFEXEC SHARE command, the LOCAL pool is scanned for matching variable names. Every match is recorded and at the end of the EXEC, the values for all these matches are gathered and transferred back to the calling EXEC (AOEXEC command). For IMFEXEC SHARE to work, at least one variable has to be specified on the AOEXEC SELECT SHARE() statement.

For example 'IMFEXEC SHARE (A B C)' causes the EXEC to determine whether variables A, B, or C currently exist in the LOCAL pool. If these variables do not exist, NO data is transferred back to the EXEC. If they exist at the time of the command, they are recorded.

Once the EXEC terminates, their values are gathered and set in the function pool of the invoking AOEXEC EXEC. This means that an 'IMFEXEC SHARE(*)' will cause all current variables in the LOCAL pool to be recorded and at EXEC termination, transferred back to the invoking EXEC.

Condition codes are listed in the following table.

Value	Description
12	This EXEC is not running under AOEXEC control. Therefore the IMFEXEC SHARE statement is inapplicable.
16	Syntax error

Example

The pattern match happens at the time of the IMFEXEC SHARE statement. If at the time the statement executes and no matches are found in the LOCAL pool but subsequently new variables are created that would have matched these variables, nothing is transferred back. For example:

```
/* REXX */
A=1
B=2
"IMFEXEC VPUT (A B) LOCAL"
"IMFEXEC SHARE (C)"
C=3
"IMFEXEC VPUT (C) LOCAL"
```

In this example, the variable C is NOT returned to the invoking EXEC.

The following sample code shows how the variable C is returned to the invoking EXEC:

```
/* REXX */
A=1
B=C
C=3
"IMFEXEC VPUT (A B C) LOCAL"
"IMFEXEC SHARE (C)"
```

STDTIME

This command instructs the Elan Workstation to use its modem to obtain the current Universal Coordinated Time (UCT). The current UCT, as well as the local date and time, are returned as variables. Refer to the AutoOPERATOR *Elan Administration Guide* for more information.

Command	Parameters
STDTIME	Values can be: Var1 Var2 Var3 Var4

The following table describes the parameters.

Parameter	Function	Notes
Var1	Name of variable to receive the GMT date	
Var2	Name of variable to receive the GMT time	
Var3	Name of variable to receive the local date	
Var4	Name of variable to receive the local time	

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
4	A variable name is invalid
8	The request timed out
12	Elan could not execute the request
16	Elan communications were not established

Example

This example sets the system clock using the local time and date obtained by the Elan Workstation.

```
/* REXX */
"IMFEXEC STDTIME V1 V2 LDATE LCLOCK"
"IMFEXEC CMD #SET DATE="LDATE",CLOCK="LCLOCK
```

SUBMIT

This command submits a job from a data set to MVS for execution in the background. Installation SUBMIT exits are honored.

The user ID associated with the submitted job is the value specified with the PREFIX= parameter of BBPPARM member AAOEXP00.

Command	Parameters
SUBMIT	'Data set name'

The following table describes the parameters.

Parameter	Function	Notes
Data set name	The data set name or name of a member of a partitioned data set that defines a JCL stream	A member name of * is not supported. Job cards cannot be automatically created and must be supplied within the user submitted jobstream.

The following TSO SUBMIT keywords are not supported by IMFEXEC SUBMIT:

- HOLD
- NOHOLD
- NOTIFY
- NONOTIFY
- PASSWORD
- NOPASSWORD
- USER( )
- NOUSER

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
16	Error, job was not submitted

Example

This example submits the JCL stream DBREC010 for execution.

```
/* REXX */
"IMFEXEC SUBMIT 'SYSP.PROD.JOBS(DBREC010)'"
```

Note: The IMFEXEC SUBMIT command works only if it is issued after JES is started.

TAILOR

Using skeleton tailoring, you can manipulate the contents of members of partitioned data sets and REXX or TSO CLIST variables. Skeleton tailoring reads the member line by line (or examines the variables) while substituting variable indicators within these lines or variables. At the same time, it follows a number of directives to generate a member of a partitioned data set (or a set of output variables).

Skeleton tailoring can be used for a number of purposes including JCL tailoring. Since its output can be a set of variables and thesevariables can be directly submitted by an IMFEXEC JESSUBM command, you need to review the feasibility of using the IMFEXEC TAILOR and IMFEXEC JESSUBM command together.

Command	Parameters
TAILOR	MEMIN() | STEMIN() MEMOUT() | STEMOUT() [DD()] [INCLUDE()] [SEARCH()] [DEBUG] | NODEBUG]

The following table describes the parameters.

Parameter	Function	Notes
DD	Name of the DD statement where the MEMIN() member is read and the MEMOUT() member is written.	Can be a maximum eight characters in length and must conform to the DD name specification.
MEMIN()	Name of the PDS member to read	Can be a maximum eight characters in length and must conform to the member name specification. Either DD() and MEMIN() or STEMIN() must be specified. When MEMIN() is specified DD() must also be specified. MEMIN() and STEMIN() are mutually exclusive.
MEMOUT()	Name of the PDS member to which the output is written	Can be a maximum eight characters in length and must conform to the member name specification. MEMOUT() and STEMOUT() are mutually exclusive. Either DD() and MEMOUT() or STEMOUT() must be specified. When MEMOUT() is specified DD() must also be specified.

STEMIN()	Variable stem prefix for input lines. This prefix is followed by a period which is followed by a number. The "var.0" contains the count of variables supplied and var.1 to var.n contain the data input to the tailor function.	Can be a maximum of 26 characters in length, and must conform to variable name conventions. Either DD() and MEMIN() or STEMIN() must be specified. When MEMIN() is specified DD() must also be specified. MEMIN() and STEMIN() are mutually exclusive. The 0 index is assumed to contain the number of variables to process. If var.0 is greater than 99999, an error message will be issued, a non-zero return code set, and the last variable used will be var.99999.
INCLUDE()	Name of a DD statement from which to process)INCLUDE directives	Optional, if not specified, no substitutions are performed. Can be a maximum eight characters in length and must conform to the DD name conventions. This DD statement will be used for all directives.
SEARCH()	Search order used to satisfy variable references. This parameter defines the variable pools and the order to process them.	Optional. Possible values are TSO LOCAL SHARED PROFILE Multiple values can be specified. They can be combined or separated by commas or spaces. The search order is position dependent and can contains multiple values. If you specify SEARCH(‘’), no variable substitution is performed but the directives are interpreted. SEARCH() must include TSO if the input stream uses the)DO directive with an index specified. Long variables are not supported. The default is ‘’, which means no substitution.

STEMOUT()	Variable stem prefix for output lines. This prefix is followed by a period which is followed by a number. The "var.0" contains the count of variables created from the TAILOR function, and var.1 to var.n contain the data lines returned from the tailor function.	The result of the tailoring processing is saved into the stem variables. All existing variables are overwritten and the number of generated variables is placed into the 0 index. The maximum length is 26 characters and it must conform to the variable naming conventions. Either DD() and MEMOUT() or STEMOUT() must be specified. When MEMOUT() is specified DD() must also be specified. If more than 99999 lines are generated, an error message will be issued, a non-zero return code will be set, and the last variable generated will be var.99999.
DEBUG	Traces every line and every pass to the BBI journal.	

Condition Codes

The following table describes condition codes returned after issuing an IMFEXEC TAILOR command statement..

Value	Description
4	An error occurred while reading input for tailoring from variables or a PDS member. Messages indicate the specific condition.
8	A catastrophic error processing the input occurred.
12	An error occurred whilewriting output to variables or a PDS member.
16	A syntax error occurred while parsing parameters, such as mutually exclusive or inclusive.

Examples of Variable Substitution

In the following example, the member Recover is read from the PDS allocated with the DD statement AOJCL. Variable substitution as well as any tailoring processing is performed.When a variable is found in the input stream, the TSO pool (followed by the local pool) is searched for the purposes of substitution.

The results of the search are placed into the REXX stem variable `TEMPTLR.X.`

```
“IMFEXEC TAILOR DD(AOJCL) MEMIN(RECOVER) SEARCH(TSO LOCAL)STEMOUT(TEMPTLR)”
```

Note: The total number of output variables must not exceed 99,999.

IMFEXEC TAILOR Processing

The following commands are interpreted in the data passed to the IMFEXEC TAILOR command, either from a PDS or from in-core variables. Note that the control statements are subject to variable substitution processing.

Control Statement	Function
)INCL (membername)	Reads the contents of the member (member name) and insert it into the current skeleton tailoring process. The contents are processed using variable substitution and processing directives are interpreted. (Member name) can be a variable.
)DO (times) [(x)]	Processes the data the number of times indicated by (times). Times is a numeric constant or a variable (including stem variable). For example: `)DO VAR.0 index` The second operand (x) is optional. During processing the variable (x) is set to the current iteration, beginning with 1. This control statement enables the iterative processing of multiple variables. The end of the processing loop is indicated by an **)**END directive or it is assumed at the end of the input stream. Nested loops can be used. When using [(x)], the SEARCH() keyword on the command must include TSO. See “Example 5” on page 372.
)END	Indicates the end of a loop.

Control Statement	Function

)SUBSTITUTION (ON) | (OFF)	Turns variable substitution ON or OFF. The)SUBSTITUTION directive can be followed by one or two parameters. The first parameter must be present and either 'ON' or 'OFF'. The second parameter (which is optional) can be a valid non-negative integer in the range from 0 to 99. It specifies the number of variable scanning passes to perform on every input line. This parameter is equivalent to the count on the DEFAULT statement. The SUBSTITUTION statement can be specified anywhere in the input stream, whereas the DEFAULT statement is only valid as the first statement in a member or stem.
	
	You can dynamically change the levels of substitution within the input stream. For example:
	
	`)SUBSTITUTION ON 16`
	`some lines1`
	`)SUBSTITUTION ON 2`
	`some lines2`
	`)SUBSTITUTION OFF`
	`some lines3`
	`)SUBSTITUTION ON`
	
	Note: The default value of ON is always the last used value. If you use OFF, then ON (with no number) as in this example, the default value of ON is 2.
	
	In the following example, the default value of ON is 16 (the last used value).
	
	`)SUBSTITUTION ON 16`
	`)SUBSTITUTION OFF`
	`)SUBSTITUTION ON`
	
	In the following example, the default value of ON is 4 (the last used value).
	
	`)SUBSTITUTION ON 16`
	`)SUBSTITUTION OFF 4`
	`)SUBSTITUTION ON`

)CM	Indicates a comment line. It will be ignored during processing.
)DEFAULT xyzz	X: Directive recognition character (default is ‘)’). Y: Variable recognition character (default ‘is &’). ZZ: The number of variables substitution passes to perform (two-digits ranging from 0 to 16). **Note:** This directive applies to the current member only. When a member is included, it temporarily overrides any previous directives in effect. No variable substitution will be performed for a line containing the this directive.

Variable Substitution

A variable is assumed when the variable recognition character is detected. This character defaults to an ampersand (&) and can be changed for each member processed through the)DEFAULT directive.

The following three rules apply when processing a line:

1. When a variable is detected in the input stream, it is replaced with its value. A variable is any string preceded by an &, which can be overridden by the)DEFAULT directive.
2. The characters that follow the & up to a blank or a slash (\ ) are assumed to be the variable name. If the variable name is followed by a \, the \ is discarded during processing.
3. The contents of a variable string beginning with two ampersand signs (&&x) are assumed to be that string less one ampersand sign (per substitution pass).

If a variable is not found, nothing is inserted and the variable instruction is discarded, which is comparable to the variable substitution in the Rules Processor.

Optionally, a variable (without intervening blanks or \) can be followed directly by one or two parameters, separated by colons. These parameters must be constants and must allow for substring processing for the variable. The first parameter indicates the beginning location in the contents of the variable, while the second one designates the length of the substring. If the length of the substring should exceed the actual length of the contents of the variable, the variable will be implicitly truncated.

Example: &VAR:2

Resolves the variable VAR, and inserts its contents beginning with the second character, into the output stream.

Example: &VAR:2:3.

Resolves the variable VAR and inserts its contents, beginning with the second character for a length of three, into the output stream. If the variable contents are shorter than four characters, substitution ends with the last character.

Examples of Variable Substitution

The following examples demonstrate processing in increasing complexity.

Example 1

The following TSO variables exist:

Variable	Contents
D1	‘JOB’
D2	‘SYSIN’

The following input stream is processed:

```
/* rexx */
in.1 = "//JOBA &D1"
in.2 = "//STEP1 EXEC PGM=&D1"
in.3 = "//&D2 DD *"
in.0 = 3
"IMFEXEC TAILOR STEMIN(IN) STEMOUT(OUT) SEARCH(TSO)"

After the TAILOR command, variable OUT.0 would contain 3.
variable OUT.1: //JOBA JOB
variable OUT.2://STEP1 EXEC PGM=JOB
variable OUT.3://SYSIN DD *
```

Example 2

The following TSO variables exist:

Variable	Contents
D1	‘JOB’
D2	‘SYSIN’

The following LOCAL variables exist:

Variable	Contents
D2	‘TEST’

The following input stream is processed:

```
//JOBA &D1
//STEP1 EXEC PGM=&D1
//&D2 DD *
```

This stream is processed with the following statement (fragmented):

```
IMFEXEC TAILOR ... SEARCH(TSO)
```

The output stream looks as follows:

```
//JOBA JOB
//STEP1 EXEC PGM=JOB
//SYSIN DD *
```

This stream is processed with the following statement (fragmented):

```
IMFEXEC TAILOR ... SEARCH(LOCAL TSO)
```

The output stream looks as follows:

```
//JOBA JOB
//STEP1 EXEC PGM=TEST
//TEST DD *
```

Example 3

The following TSO variables exist:

Variable	Contents
D2	'SYSIN'

The following input stream is processed:

```
//JOBA &D1 (3211)
//STEP1 EXEC PGM=&D1
//&D2 DD *
```

This stream is processed with the following statement (fragmented):

```
IMFEXEC TAILOR ... SEARCH(TSO)
```

The output stream looks as follows:

```
///JOBA (3211)
//STEP1 EXEC PGM=
//SYSIN DD *
```

Example 4

A member CMDBASE exists in BBPARM with the following contents:

```
COPY INDD(&INDD\) TO OUTDD(&OUTDD\)
```

The following TSO variables exist:

Variable	Contents
D1	'JOB'
D2	'SYSIN'
INDD	'SYSIN.PARMLIB'
OUTDD	'MY.PARMLIB'
INCLUDE	'CMDBASE'

The following input stream is processed:

```
//JOBA &D1
//STEP1 EXEC PGM=&D1
//&D2 DD *
)INCL &CMDBASE
```

This stream is processed with the following statement (fragment):

```
IMFEXEC TAILOR ... SEARCH(TSO)



//JOBA JOB
//STEP1 EXEC PGM=JOB
//SYSIN DD *
COPY INDD(SYS1.PARMLIB) TO OUTDD(MY.PARMLIB)
```

Example 5

The following TSO variables exist:

Variable	Contents
D1	'JOB'
D2	'SYSIN'

The following input stream is processed:

```
//JOBA &D1
//STEP1 EXEC PGM=&D1
//&D2 DD *
)DO 5
CALL PROCESS
)END
```

This stream is processed with the following statement (fragmented):

```
IMFEXEC TAILOR ... SEARCH(TSO)
```

The output stream looks as follows:

```
/JOBA JOB
//STEP1 EXEC PGM=JOB
//SYSIN DD *
CALL PROCESS
CALL PROCESS
CALL PROCESS
CALL PROCESS
CALL PROCESS
```

Example 6

The following TSO variables exist:

Variable	Contents
D1	'JOB'
D2	'SYSIN'
P1	'VOL003'

Variable	Contents
P2	‘VOL004’
P3	‘VOL005’
P4	‘VOL006’

The following LOCAL variables exist:

Variable	Contents
P5	‘STOR001’

The following input stream is processed:

```
//JOBA &D1
//STEP1 EXEC PGM=&D1
//&D2 DD *
)DO 5 WITH INDEX
CALL PROCESS &&P\&INDEX
)END
```

This stream is processed with the following statement (fragment):

```
IMFEXEC TAILOR ... SEARCH(TSO LOCAL)
```

The output stream will look as follows:

```
//JOBA JOB
//STEP1 EXEC PGM=JOB
//SYSIN DD *
CALL PROCESS VOL003
CALL PROCESS VOL004
CALL PROCESS VOL005
CALL PROCESS VOL006
CALL PROCESS STOR001
```

Explanation of DO loop: In this example, the data CALL PROCESS &&P\&INDEX is processed five times. During these iterations, the variable INDEX is set to the current iteration count. That means the statement during the first pass is substituted to

```
CALL PROCESS &P1
```

Since the contents of a variable beginning with && are assumed to equal the string minus one ampersand sign, &&P has the value of &P.

On the second pass, &P1 is substituted as VOL003.

Substitution passes stop internally when the EXEC detects a truncation of the output that exceeds 80 characters in width, or no more variables to substitute.

Note: The default is two passes, but it can be overridden using the)DEFAULT directive in the input stream.

Example 7

The following variables are used:

Variable	Contents
DSN.0=	3
DSN.1	'MY.DATASET’
DSN.2	'YOUR.DATASET’
DSN.3	‘ANOTHER.DATASET’
VOL	‘SYSDA’

The following input stream is processed:

```
//XJOB JOB
/STEP1 EXEC PGM=IEFBR14
)DO &DSN.0 INDEX
//DD&INDEX DD DISP=SHR,DSN=&&DSN\.&INDEX\\,VOL=&VOL
)END
```

A loop is generated. The statements between)DO and)END are executed as many times as the contents of the variable &DSN.0. Every time the loop is executed the variable INDEX is set to the execution count which means the first time the loop executes, it is set to 1, the next time, it will be set to 2, etc.

The statement

```
//DD&INDEX DD DISP=SHR,DSN=DSN&&DSN\.&INDEX\\,VOL=&VOL
```

will execute three times, with the value of the variable INDEX varying from 1 to 3. The)DO statement is translated to

```
)DO 3 INDEX
```

Explanation of the looped statement

On the first pass it is translated to

```
//DD1 DD DISP=SHR,DSN=&DSN.1\,VOL=SYSDA
```

Note that the first \ following `&INDEX\\,VOL=&VOL` has been discarded according to the rules.

Now the second pass translates it again:

```
//DD1 DD DISP=SHR,DSN=MY.DATASET,VOL=SYSDA
```

The same substitution is performed on the second and third iteration. See “Variable Substitution” on page 368 for variable processing rules.

When &&XYZ\\ is specified, rule #1 is followed where the name of the variable is defined by looking at the string following the first & sign, until a blank or \ is detected. Therefore, the actual name of this variable would be &XYZ. This variable can never exist in REXX, nor in CLIST.

However, following rule number 3, the value of this variable is assumed to be the name of the variable. Therefore the contents of &XYZ actually is &XYZ. The string &&XYZ\\ became &XYZ\ on the first pass.

On the second pass, &XYZ\ is the name of the variable for it is again the string following the & sign up to the next blank or \. The name of the variable becomes XYZ. This variable name is valid and its contents are substituted.

See “Variable Substitution” on page 368 for variable processing rules.

TRANSMIT

This command responds in an OSPI session to a VTAM application by sending a 3270 input data stream. This is equivalent to an operator pressing an active (non-local) key on a 3270 terminal keyboard.

Refer to “Interacting with VTAM-Applications with OSPI” on page 113 for more information about using this command and OSPI.

When using this statement, you must remember to code IMFEXEC with the TRANSMIT command. If you do not, you might cause the TSO/E TRANSMIT command to be invoked.

Command	Parameters
TRANSMIT	[ENTER|CLEAR|PFx|PAx] SESSION(Session identifier)

The following table describes the parameters.

Parameter	Function	Notes
Keystroke	The key used to transmit the buffer	One of the following pools: • ENTER • CLEAR • PFx (where x = 1-24) • PAx (where x= 1-3)
SESSION	Session identifier for session to reference	This session identifier is returned initially through the LOGON command.

After a TRANSMIT command, a receive function is implied. After the receive, the virtual screen buffer is modified with the application's data.

You can query the buffer by coding the VGET command for any of the session variables or by using the SCAN command. Some applications require the explicit use of a RECEIVE command after a TRANSMIT. Refer to “Interacting with VTAM-Applications with OSPI” on page 113 for more information.

Condition codes are listed in the following table.

Value	Description
0	Command responded before wait time expired
8	Session not found

Example

This example command transmits the modified virtual screen back to the host application using the ENTER key. The SESSION keyword designates the referred session.

```
/* REXX */
"IMFEXEC TRANSMIT ENTER SESSION("OSISESS")"
```

TYPE

This command enters data into the virtual screen image maintained by an OSPI session.

Refer to “Interacting with VTAM-Applications with OSPI” on page 113 for more information about using this command and OSPI.

Command	Parameters
TYPE	[TAB|BACKTAB|ERASEEOF|HOME|RESET] [ROW(Row)] [COL(Column)] [TEXT(Text)] SESSION(Session identifier)

The following table describes the parameters.

Parameter	Function	Notes
Keystroke	A local 3270 function key to type before entering the text	One of the following: • TAB • BACKTAB • ERASEEOF • HOME • RESET
ROW	The screen row at which to enter the data	Numeric value in the range: 1 - (minus) the maximum number of rows emulated by the current terminal type.
COL	The screen column at which to enter the data	Numeric value in the range 1-80.
TEXT	The text to be entered on the screen	Maximum length is 255 characters.
SESSION	Session identifier for session to reference	This session identifier is returned initially through the LOGON command.

This command does not transmit any data to the host application. The TRANSMIT command passes the modified virtual screen buffer back to the application.

Condition codes are listed in the following table.

Value	Description
0	Command responded before wait time expired
4	Timeout occurred
8	Session not found

Example

This example command tabs to the next input field on the virtual screen before entering the text. The session addressed by this command is contained in the variable OSISESS.

```
/* REXX */
"IMFEXEC TYPE TAB TEXT('CATALOG') SESSION("OSISESS")"
```

VCKP

This command writes updated profile pool variables to the BBIVARS data set.

Command	Parameters
VCKP	This command has no parameters.

Checkpoints for PROFILE variables are taken automatically at EXEC termination if these variables were updated in the EXEC. With VCKP, you can write variables that were updated in the profile pool directly to the BBIVARS data set on disk at any point in time during EXEC execution.

This is recommended if the EXEC does not terminate for extended periods of dime. Use this when you need to guarantee the integrity of certain variables..

Important Note

Presence of the VCKP command within an EXEC's loop might degrade the performance of the BBI-SS PAS.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
20	Invalid syntax used

Example

This example saves the variables IMSSTART, IMSTIME, and IMSCHKPT in the profile pool. It forces writing these variables to the BBIVARS data set since the EXEC does not terminate immediately but branches to the label LAB01 (not shown).

```
/* REXX */
"IMFEXEC VPUT (IMSSTAT IMSTIME IMSCHKPT) PROFILE"
"IMFEXEC VCKP"
```

VDCL

This command equates a variable to a list of TSO variables which are automatically parsed and combined during VGET and VPUT operations.

Command	Parameters
VDCL	List name LIST(v1 ... vn)

The following table describes the parameters.

Parameter	Function	Notes
List name	The name of a list of LOCAL or GLOBAL variables	
LIST	The names of the TSO variables to be used in the EXEC	The maximum number of each variable name is 32 characters. The total length of the list of TSO variables cannot exceed 255. The maximum number of variables in the list is 99. Using this statement redundantly in an EXEC will slow the EXEC's execution.

You cannot directly access a LIST variable but instead, you must reference the individual subcomponents. This command is used in conjunction with the IMFEXEC VGET INTO() command and allows for simplified parsing of character strings. There is no corresponding command to reverse the effect of a VDCL.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
12	Invalid syntax; no variables were passed

Example

Refer to the List name in the VGET command but refer to the actual variable names when you want to access the data in the list. This command is used in conjunction with the IMFEXEC VGET INTO() command and allows for simplified parsing of character strings. There is no corresponding command to reverse the effect of a VDCL.

```
/* REXX */
"IMFEXEC VDCL LINE LIST(P1 P2 P3 P4 P5)"
VAR='THIS IS A TEST'
"IMFEXEC VPUT VAR LOCAL"
"IMFEXEC VGET VAR INTO(LINE) LOCAL"
"IMFEXEC MSG '"P1 P2 P3 P4 P5"'"
```

After execution of these example commands, the contents of the variables are:

```
THIS IS A TEST .
```

Note: A list of 99 variables with a total of 256 characters will generate the error message: `MORE THAN 99 VARIABLES`.

VDEL

This command deletes one or more variables from one of the AutoOPERATOR variable pools.

Command	Parameters
VDEL	Variable name |pattern|(v1 ... vn) [LOCAL|SHARED|PROFILE] TARGET(ssid)

The following table describes the parameters.

Parameter	Function	Notes

Variable name |pattern|(v1 ... vn)	The name of one or more AutoOPERATOR variables	If more than one variable is specified, the variable names must be enclosed in parentheses.
		
		The maximum length of this parameter is 252 bytes. All variables in a pool can be deleted by using the identifier ALL instead of naming all variables individually. A variable cannot begin with a numeric nor can it contain special characters.
		
		An example of using a pattern is:
		
		`IMFEXEC VDEL CICS*`
		
		The variable names can be generically expressed by using an asterisk. However, the VDEL command statement assumes the presence of an asterisk means the end of the string.
		
		`IMFEXEC VDEL ABC*D`
		
		is treated as if you coded:
		
		`IMFEXEC VDEL ABC*`
		
		In addition, if you try to use an asterisk within a string of text, you will receive a return code for invalid syntax usage. For example, if you try to issue a pattern:
		
		`IMFEXEC VDEL CSM*MSG12`
		
		you will receive a return code of IMFCC=16 (for invalid syntax usage).

Parameter	Function	Notes
Pool identifier	The pool in which the designated variables reside	One of the following pools: • LOCAL • SHARED • PROFILE
TARGET	Allows you specify the BBI-SS PAS ID of another BBI-SS PAS. You can then VDEL variables from one BBI-SS PAS to another BBI-SS PAS that communicates with it.	The TARGET keyword can be used with IMFEXEC commands VDEL, VGET, and VPUT.

Note: This command does not affect variables that have already been retrieved from one of the pools.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
8	One of the following conditions is true: • Variable does not exist • Node not found in BBPARM member BBINOD00 (when TARGET is used)
12	TARGET not found in BBPARM member BBIJNT00
16	Syntax error; TARGET not found in BBPARM member BBIJNT00
20	One of the following conditions is true: • Severe error (internal) and pool was not found • Variable overflow When using the TARGET keyword to VPUT a variable to another target, there is a limit of (approximately) 7000 bytes of data that can be sent to another target.

Example

This example deletes all variables ending in the characters TEST from the shared variable pool. It uses the VLST command to retrieve all variable names.

```
/* REXX */
"IMFEXEC VLST * SHARED"

DO I = 1 TO IMFNOL
  "IMFEXEC VGET LINE"N "LOCAL"
  "IMFEXEC VGET" LINE||N" INTO(DUMMY1) SHARED"
  LEN = LENGTH(VALUE('LINE'I))
  IF LEN > 3 THEN
    IF SUBSTR(VALUE('LINE'I),LEN-3,LEN) = 'TEST' THEN
    "IMFEXEC VDEL" VALUE('LINE'I) "SHARED"
END
```

VDELL

This command deletes one or more long variables from one of the AutoOPERATOR variable pools.

Note: This variable operation only supports a subset of the functions available for the short variables. It ONLY affects and searches for long variables. If a short variable (created with VPUT instead of VPUTL) with the specified name exists, it is ignored.

Command	Parameters
VDELL	[LOCAL|SHARED|PROFILE)] Variable name|pattern|(v1 ... vn)

The following table describes the parameters.

Parameter	Function	Notes
Pool identifier	The pool in which the designated variables reside	One of the following pools: • LOCAL • SHARED • PROFILE SHARED is the default.

Variable name| pattern|(v1 ... vn)	The name of one or more variables or a pattern.	If more than one variable is specified, the variable names must be enclosed in parentheses.
		
		The maximum length of this parameter is 252 bytes. All variables in a pool can be deleted by using the identifier ALL instead of naming all variables individually. A variable cannot begin with a numeric nor can it contain special characters.
		
		An example of using a pattern is:
		
		`IMFEXEC VDELL CICS*`
		
		The variable names can be generically expressed by using an asterisk. However, the VDEL command statement assumes the presence of an asterisk means the end of the string.
		
		`IMFEXEC VDELL ABC*D`
		
		is treated as if you coded:
		
		`IMFEXEC VDELL ABC*`
		
		In addition, if you try to use an asterisk within a string of text, you will receive a return code for invalid syntax usage. For example, if you try to issue a pattern:
		
		`IMFEXEC VDELL CSM*MSG12`
		
		you will receive a return code of IMFCC=16 (for invalid syntax usage).

Note: This command does not affect variables that have already been retrieved from one of the pools.

Condition codes are listed in the following table.

Value	Description
0	The variable existed in the target pool and has been deleted.
8	No long variable with this name has been found in the target pool.
12	Attempt to delete a read-only variable (for example, Q-type variable was specified which cannot be deleted with VDELL).
16	Syntax error.
20	Variable pool not found (BBIVARS not allocated)

Example

The PROFILE pool is searched for a long variable with the name of X. If found it is deleted.

```
"IMFEXEC VDELL X PROFILE"
```

VDEQ

This command issues an MVS Dequeue for a major name of BBIUSER.

Command	Parameters
VDEQ	'Symbolic name'

The following table describes the parameters.

Parameter	Function	Notes
'Symbolic name'	The minor name of the Dequeue	1-255 alphanumeric characters.

The VDEQ command returns without errors if the enqueue was already freed by VDEQ issued in the same EXEC. Use this command in conjunction with the VENQ command.

Condition codes and the corresponding return codes (listed in the variable IMFRC) are listed in the following table. The IMFRC value represents the return code

IMFCC Value	Description	IMFRC Value	Description
0	Enqueue released; no warning applies	0	Enqueue release
4	Enqueue not held or is already released; warning applies	8	Enqueue not held or already released
16	Syntax error	N/A	N/A

Example

This example releases the enqueue on the symbolic resource STARTUP. Other EXECs waiting for this resource resume processing.

```
/* REXX */
"IMFEXEC VDEQ 'STARTUP'"
```

VENQ

This command issues an MVS Enqueue for a major name of BBIUSER. It establishes a shared or an exclusive ENQUEUE for the given parameter.

Command	Parameters
VENQ	'Symbolic name' Disposition TEST

Use this command whenever access to a particular resource needs to be serialized.

The following table describes the parameters.

Parameter	Function	Notes
'Symbolic name'	The minor name of the Enqueue	1-255 alphanumeric characters.
Disposition	Type of Enqueue to issue	Either SHR or EXC. SHR means the resource can be shared between tasks in the same address space. EXC means a task has an exclusive enqueue and no other tasks can enqueue at that resource.
TEST	Specifies that no ENQ is obtained but the availability of an ENQ will be tested	A different set of condition codes is returned if this parameter is used. Refer to the condition code tables below.

The VENQ command returns without errors if the enqueue was already obtained for a previous VENQ issued in the same EXEC.

Condition codes and the corresponding return codes (listed in the variable IMFRC) are listed in the following table. The IMFRC value represents the return code returned from the actual MVS enqueue macro.

IMFCC Value	Description	IMFRC Value	Description
0	Enqueue received; no warning applies	0	Enqueue obtained or is obtainable
4	Enqueue already held; warning applies	8	EXEC already has control of the enqueue
8	Enqueue not obtained; warning applies	14	Previous request for enqueue has been made for the same task; the EXEC does not have control of the enqueue
8	Enqueue not obtained; warning applies	18	Limit for concurrent requests reached
16	Syntax error	N/A	N/A

If you use the parameter TEST with the IMFEXEC ENQUEUE statement, a different set of condition codes and the corresponding return codes (listed in the variable

IMFCC Value	Description	IMFRC Value	Description
0	Enqueue obtainable; no warning applies	0	Enqueue obtained or is obtainable
4	Enqueue already held; warning applies	8	EXEC already has control of the enqueue
8	Enqueue not obtainable; error warning applies	4	Resource not available
8	Enqueue not obtainable; error warning applies	14	Previous request for enqueue has been made for the same task; the EXEC does not have control of the enqueue
16	Syntax error	N/A	N/A

Examples

This section contains examples using the IMFEXEC VENQ command statement. A brief discussion follows each example.

Example 1:

```
/* REXX */
"IMFEXEC VENQ 'STARTUP' SHR"
```

This example command establishes a shared ENQUEUE for the name STARTUP.

Example 2:

```
"IMFEXEC VENQ 'PRODCICS' SHR TEST"
```

This example command tests whether the current EXEC can obtain shared access to the resource PRODCICS.

VGET

This command copies one or more variables from one of the AutoOPERATOR pools into the EXECs function pool.

Command	Parameters
VGET	Variable name|(v1 ... vn) [INTO(Variable)] [LOCAL|SHARED|PROFILE] [DECRYPT(xyz)] DELIM(',') TARGET(ssid)

The following table describes the parameters.

Parameter	Function	Notes
Variable name|(v1 ... vn)	The name of one or more variables to copy	If more than one variable is specified, the variable names must be enclosed in parentheses. Each variable name can be up to 32 characters. The maximum length of the combined variable values is 252 bytes.
INTO	An optional keyword that you can use in conjunction with the VDCL command to map a string of characters into a list of individual variables	The variable to receive the values should have been declared with the IMFEXEC VDCL statement.
Pool identifier	The pool in which the designated variables reside	One of the following pools: • LOCAL • SHARED • PROFILE

Parameter	Function	Notes
DECRYPT(xyz)	Specifies a character string that can be used for decrypting variable contents	This parameter **must** be used in conjunction with the ENCRYPT parameter on an IMFEXEC VPUT command at the same time. If this is not done, the contents of the data will not match. The character string can be 2-255 characters long. Enclose the character string in single quote marks if the string contains blanks. Only individual variables can be decrypted. List variables cannot be decrypted. Refer to “VPUT” on page 399 for information about the ENCRYPT parameter.
DELIM(',')	Allows you to specify characters (instead of blanks) to delimit words or characters into separate variables	Blank is the default.
TARGET	Allows you specify the BBI-SS PAS ID of another BBI-SS PAS. You can then VGET variables from one BBI-SS PAS to another BBI-SS PAS that communicates with it.	The TARGET keyword can be used with IMFEXEC commands VDEL, VGET, and VPUT.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
8	One of the following is true: • Variable does not exist • Node not found in BBPARM member BBINOD00 (when TARGET is used)
12	TARGET not found in BBPARM member BBIJNT00
16	Node not available
20	One of the following conditions is true: • Severe error (internal) and pool was not found • Variable overflow When using the TARGET keyword to VPUT a variable to another target, there is a limit of (approximately) 7000 bytes of data that can be sent to another target. This includes both the variable name and variable value.

Examples

This section contains examples using the IMFEXEC VGET command statement. A brief discussion follows each example.

Example 1

```
/* REXX */
"IMFEXEC VLST * SHARED"

DO I = 1 TO IMFNOL
  "IMFEXEC VGET LINE"I "LOCAL"
  "IMFEXEC VGET" VALUE('LINE'I) "SHARED"
  "IMFEXEC MSG '.."VALUE(VALUE('LINE'I))"'"
END
```

This example displays the contents of the EXECs SHARED variable pool. It uses the VLST command to retrieve the names of all variables in that pool.

It then VGETs them one after the other and displays their contents.

Example 2

```
/* REXX */
"IMFEXEC VDCL CICSL20 LIST(V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12)"
"IMFEXEC VGET LINE20 INTO(CICSL20) LOCAL"
```

The VDCL command specifies that CICSL20 is a list of 12 variables, V1 to V12. The VGET command maps the data returned for LINE20 into the 12 variables. The local variables, V1 through V12, can now be processed by other commands within the EXEC.

Example 3

```
/* REXX */
ABC=SUBSTR(THIS IS A DATA ENCRYPTION EXAMPLE)
"IMFEXEC VPUT ABC SHARED ENCRYPT('DATASTREAM')"
"IMFEXEC VGET ABC SHARED DECRYPT('DATASTREAM')"
```

This is an example of how to use the IMFEXEC VPUT ENCRYPT and IMFEXEC VGET DECRYPT parameters. Notice how both parameters specify `('DATASTREAM')`. This is done to ensure that when the variable ABC is decrypted, the contents of the variable will be accurate.

VGETL

This command copies one or more long variables from one of the AutoOPERATOR pools into the TSO pool.

Note: This variable operation only supports a subset of the functions available for the short variables. It ONLY affects and searches for long variables. If a short variable (created with VPUT instead of VPUTL) with the specified name exists it is ignored.

Command	Parameters
VGETL	[LOCAL|SHARED|PROFILE)] Variable name|(v1 ... vn)

The following table describes the parameters.

Parameter	Function	Notes
Pool identifier	The pool in which the designated variables reside	One of the following pools: • LOCAL • SHARED • PROFILE SHARED is the default.
Variable name|(v1 ... vn)	The name of one or more variables	Required parameter. If more than one variable is specified, the variable names must be enclosed in parentheses. Each variable name can be up to 30 characters. The maximum length of the combined variable values is 252 bytes

Condition codes are listed in the following table.

Value	Description
0	The variable existed in the target pool and has been retrieved.
8	No long variable with this name has been found in the target pool.
16	Syntax error
20	Variable pool not found (BBIVARS not allocated)

Examples

The PROFILE pool is searched for a long variable with the name of X. If found it is placed into the TSO pool. It is then assigned to the variable Y.

```
"IMFEXEC VGETL X PROFILE"
y=x
```

VLST

This command lists variable names defined in the AutoOPERATOR pools. It returns those names in LOCAL variables LINE1 through LINEn and sets IMFNOL to the count of LINEs.

Command	Parameters
VLST	Variable pattern [SHARED | PROFILE | REXX]

The following table describes the parameters.

Parameter	Function	Notes
Variable pattern	The name or name pattern for specifying variable names to retrieve	The variable names can be generically expressed by using an asterisk.
Pool identifier	The pool in which the designated variables reside	One of the following pools: • SHARED • PROFILE • REXX

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
8	No variable was found
12	Variable pool is not available

Example

The following EXEC uses the IMFEXEC VLST command to retrieve all the variables that begin with RETRY and then reports the number of retries. The variable RETRY.termname will contain the number of retries for a terminal.

```
/* REXX */
/*********************************************************/
/* THIS EXEC WILL PRINT RETRY COUNTS FOR ALL TERMINALS   */
/*********************************************************/

   "IMFEXEC VLST RETRY* SHARED"
    IF IMFCC > 0 THEN EXIT

    "IMFEXEC VDCL DUMMY1 LIST(VARNAME)"
    "IMFEXEC VDCL DUMMY2 LIST(DATE COUNT)"

    DO N = 1 TO IMFNOL
       "IMFEXEC VGET LINE"N "INTO(DUMMY1) SHARED"
       "IMFEXEC VGET" VALUE(VARNAME) "INTO(DUMMY2) SHARED"
       END = LENGTH(VARNAME)
       NOD = SUBSTR(VARNAME,7,END-7)
       "IMFEXEC MSG '*TERMINAL:" NOD "RETRIES: "COUNT"'"
   END
   "IMFEXEC VDEL RETRY* SHARED"
```

VLSTL

This command retrieves a long variable from the specified pool and places it intot he TSO pool.

Note: This variable operation only supports a subset of the functions available for the short variables. It ONLY affects and searches for long variables. If a short variable (created with VPUT instead of VPUTL) with the specified name exists it is ignored.

Command	Parameters
VLSTL	[SHARED|PROFILE] Variable pattern

The following table describes the parameters.

Parameter	Function	Notes
Pool identifier	The pool in which the designated variables reside	One of the following pools: • SHARED • PROFILE SHARED is the default.
Variable pattern	The name or name pattern for specifying variable names to retrieve	Required parameter. Only one variable can specified and the name must be enclosed in parentheses. Each variable name can be up to 30 characters. The variable name can be a pattern: `(A+B*)` where the following wildcards are supported: + **(plus sign)** Matches any one character. *** (asterisk)** Matches zero to any number of characters.

Condition codes are listed in the following table.

Value	Description
0	At least one variable has been found.
8	No long variable with this name has been found in the target pool.

16	Syntax error
20	Variable pool not found (BBIVARS not allocated)

Example

This EXEC lists all long variables in the SHARED pool and writes their names to the terminal.

```
/* REXX */
"IMFEXEC VLSTL * SHARED"
say IMFNOL
do i=1 to IMFNOL
  say value('LINE'I)
end
```

VPUT

This command copies one or more variables from the EXECs function pool into one of the AutoOPERATOR pools.

Command	Parameters
VPUT	Variable name|(v1 ... vn) [FROM(Variable name)] [LOCAL|SHARED|PROFILE] [USING(v1 ... vn)] [ENCRYPT(xyz)] TARGET(ssid)

The following table describes the parameters.

Parameter	Function	Notes
Variable name|(v1 ... vn)	The names of one or more variables to copy	If more than one variable is specified, the variable names must be enclosed in parentheses. Each variable name can be up to 32 characters. The maximum length of the combined variable values is 252 bytes.
FROM	An optional keyword that you can use in conjunction with the VDCL command to map a string of characters into a list of individual variables	The list is created by IMFEXEC VDCL.
Pool identifier	The pool to which the designated variables should be placed	One of the following pools: • LOCAL • SHARED • PROFILE
USING	An optional keyword that, when used with AutoOPERATOR variables, allows you to set the AutoOPERATOR variables from the LOCAL variable pool	For example: `IMFEXEC VPUT (A B C)USING (X Y Z)` allows the AutoOPERATOR variables A, B, and C be set to the TSO variable contained in X, Y, and Z.

Parameter	Function	Notes
ENCRYPT(xyz)	Specifies a character string that can be used for encrypting variable contents	This parameter **must** be used in conjunction with the DECRYPT parameter on an IMFEXEC VGET command at the same time. If this is not done, the contents of the data will not match. The character string can be 2-255 characters long. Enclose the character string in single quote marks if the string contains blanks. Only individual variables can be encrypted. List variables cannot be encrypted. Refer to “VGET” on page 391 for information about the DECRYPT parameter.
TARGET	Allows you specify the BBI-SS PAS ID of another BBI-SS PAS. You can then VPUT variables from one BBI-SS PAS to another BBI-SS PAS that communicates with it.	The TARGET keyword can be used with IMFEXEC commands VDEL, VGET, and VPUT.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
4	Variable did not previously exist in the designated pool
8	One of the following is true: • Invalid syntax used • When VPUTting a variable to the PROFILE pool, the PROFILE pool data set BBIVARS is full • Node not found in BBPARM member BBINOD00 (when TARGET is used)
12	One of the following is true: • Q-type variable was specified and cannot be VPUT • TARGET not defined in BBPARM member BBIJNT00

Value	Description
16	One of the following is true: • Internal error • Node not available and TARGET is used
20	One of the following conditions is true: • No variables in list • Variable name is invalid • Variable overflow When using the TARGET keyword to VPUT a variable to another target, there is a limit of (approximately) 7000 bytes of data that can be sent to another target. This includes both the variable name and variable value.

Examples

This section contains examples using the IMFEXEC VPUT command statement. A brief discussion follows each example.

Example 1

```
/* REXX */
"IMFEXEC VPUT (ABENDS ABENDCOUNT ABENDREASON)"
```

This example command saves the current value of `ABENDS`, `ABENDCOUNT`, and `ABENDREASON` in the shared pool.

Example 2

```
/* REXX */
"IMFEXEC VPUT ABENDS LOCAL"
```

This example command saves the current value of `ABENDS` in the local pool.

Example 3

```
/* REXX */
ABC=SUBSTR(THIS IS A DATA ENCRYPTION EXAMPLE)
"IMFEXEC VPUT ABC SHARED ENCRYPT('DATASTREAM')"
"IMFEXEC VGET ABC SHARED DECRYPT('DATASTREAM')"
```

This is an example of how to use the IMFEXEC VPUT ENCRYPT and IMFEXEC VGET DECRYPT parameters. Notice how both parameters specify `('DATASTREAM')`. This is done to ensure that when the variable ABC is decrypted, the contents of the variable will be accurate.

VPUTL

This command creates a or sets a long variable from a variable in the TSO pool.

Note: This variable operation only supports a subset of the functions available for the short variables. For example, no target system functionality is provided. It ONLY affects and searches for long variables. If a short variable (created with VPUT instead of VPUTL) with the specified name exists it is ignored.

Command	Parameters
VPUTL	[LOCAL|SHARED|PROFILE)] Variable name|(v1 ... vn)

The following table describes the parameters.

Parameter	Function	Notes
Pool identifier	The pool in which the designated variables reside	One of the following pools: • LOCAL • SHARED • PROFILE SHARED is the default.
Variable name|(v1 ... vn)	The name of one or more variables	Required parameter. Each variable name can be up to 30 characters. The maximum length of the combined variable values is 252 bytes. Variables beginning with the character Q are reserved for system variables and may not be modified.

Condition codes are listed in the following table.

Value	Description
0	The variable existed in the target pool and has been overwritten.
4	The variable did not exist in the pool and has been created.
8	Error during operation. Possible Out-of-Space condition for the PROFILE pool.
12	Attempt to set a read-only variable (for example, Q-type variable was specified which cannot be VPUT).
16	Syntax error
20	Variable pool not found. BBIVARS not allocated.

Examples

```
a='This is a test'
"IMFEXEC VPUTL A SHARED"
```

WAIT

This command suspends EXEC execution for a specified interval or until a value (or name) is posted by another EXEC using the IMFEXEC POST command statement.

Command	Parameters
WAIT	n
NAME	(name)

The following table describes the parameters.

Parameter	Function	Notes
Interval	Number of seconds to suspend execution	Numeric value in the range 1-9999.
NAME	Use this parameter with the NAME parameter in the IMFEXEC POST command statement	Can be 1-32 alphanumeric characters long. This parameter allows you to halt execution of the EXEC until either the wait time expires or until the NAME parameter in an IMFEXEC POST command is posted. Refer to “POST” on page 344 for additional information.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully. (If WAIT on NAME and INTERVAL did not expire.)
8	Attempted WAIT on NAME timed out.
16	Syntax Error

Example

This example command pauses the EXEC for 15 seconds or until the name TOKEN is posted by the IMFEXEC POST command statement in another EXEC. The EXEC processing thread remains in use.

```
/* REXX */
"IMFEXEC WAIT 15 NAME(TOKEN)"
```

WAITLIST

This command returns the IDs of EXECs in WAIT mode to the LOCAL pool in variables EXEC1 through EXECxxx. In addition:

- The variables LINE1 through LINExxx contain the names of the resources and the variable IMFNOL contains the number of lines returned
- The variables NAME1 through NAMExxx contain the names of EXECs
- The variables DATE1 through DATExxx contain the date when an EXEC started
- The variables TIME1 through TIMExxx contain the time they started

Command	Parameters
WAITLIST	pattern

The following table describes the parameters.

Parameter	Function	Notes
pattern	Is the resource name that is being waited on This parameter is not optional. You can use wildcard characters (such as * or +) where an asterisk represents one or more characters and a plus sign represents a single character.	1-32 alphanumeric characters. LINE1 through LINExxx and EXEC1 through EXECxxx are LOCAL pool variables and IMFNOL is a TSO variable. IMFNOL is valid only if the return code is 0. The variables NAME1 through NAMExxx contain the names of EXECs. The variables DATE1 through DATExxx contain the date when an EXEC started. The variables TIME1 through TIMExxx contain the time the EXEC(s) started.

Condition codes are listed in the following table.

Value	Description
0	The names of one or more waiting EXECs and associated resources were returned
4	No waiting EXECs were returned
8	The parameter was not specified
12	Syntax error

Example

This example issues the WAITLIST command to display the names of the EXECs that are awaiting an IMFEXEC POST. The names of the EXECs, dates and times are displayed in the BBI-SS PAS Journal.

```
/* REXX */
"IMFEXEC SELECT EXEC(WAIT MOO)WAIT(NO)"
"IMFEXEC SELECT EXEC(WAIT GAV)WAIT(NO)"
"IMFEXEC SELECT EXEC(WAIT MEW)WAIT(NO)"
"IMFEXEC SELECT EXEC(WAIT KWA)WAIT(NO)"
"IMFEXEC WAIT 3"
"IMFEXEC WAITLIST *"
"IMFEXEC MSG 'WTEST.'"
"IMFEXEC MSG 'WTEST.DATE:" date() " TIME:" time()"'"
"IMFEXEC MSG 'WTEST.IMFCC  = "IMFCC "'"
"IMFEXEC MSG 'WTEST.IMFRC  = "IMFRC "'"
"IMFEXEC MSG 'WTEST.IMFNOL =" imfnol "'"
"IMFEXEC MSG 'WTEST.RESOURCE",
       "  EXEC ID   NAME      DATE      TIME STARTED'"
do n = 1 to imfnol
  "IMFEXEC VGET LINE"n "LOCAL"
  "IMFEXEC VGET EXEC"n "LOCAL"
  "IMFEXEC VGET NAME"n "LOCAL"
  "IMFEXEC VGET DATE"n "LOCAL"
  "IMFEXEC VGET TIME"n "LOCAL"
  r1 = value("LINE"n);r = left(r1,10)
  r1 = value("EXEC"n);r = r left(r1,9)
  r1 = value("NAME"n);r = r left(r1,9)
  r1 = value("DATE"n);r = r left(r1,9)
  r1 = value("TIME"n);r = r left(r1,9)
  "IMFEXEC MSG 'WTEST."r"'"
end
"IMFEXEC POST 'MOO'"
"IMFEXEC POST 'GAV'"
"IMFEXEC POST 'MEW'"
"IMFEXEC POST 'KWA'"
"IMFEXEC MSG 'WTEST.'"
return
```

WTO

This command sends a message to one or more system consoles.

Command	Parameters
WTO	'Msgtext' [JOBID(n)] [ROUTCDE(n1 ... nn)] [DESC(n1 ... nn)] [CONSOLE|CN(n)] [SSID(YES|NO)] [NAME(x)]

The following table describes the parameters.

Parameter	Function	Notes
Msgtext	The text of the message to send	Maximum length is 126 characters with SSID defaulting to NO; if SSID is YES, maximum length is reduced to 119.
JOBID	Job identifier to place in SYSLOG as message issuer	1-8 characters alphanumeric, first character alpha.
ROUTCDE	Routing codes to associate with the message	Refer to the IBM publication, *Routing and Descriptor Codes*, for more information.
DESC	Descriptor codes to associate with the message	Refer to the IBM publication, *Routing and Descriptor Codes*, for more information.
CONSOLE CN	Specific Console ID of console to receive the message	Numeric identifier. You can specify NAME() or CONSOLE(), but you cannot specify both. If you omit the CONSOLE|CN and NAME keywords, the system uses the routing code specified on the ROUTCODE keyword on the DEFAULT statement in the CONSOLxx member of SYS1.PARMLIB.
SSID	Appends the subsystem ID to the end of the message	Either YES or NO.
NAME	A valid console name to where the message is sent	You can specify NAME() or CONSOLE(), but you cannot specify both. If you omit the CONSOLE|CN and NAME keywords, the system uses the routing code specified on the ROUTCODE keyword on the DEFAULT statement in the CONSOLxx member of SYS1.PARMLIB.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully. The variable IMFWTDOM contains the DOM ID for this WTO. You may need to use this value in a later invocation of the IMFEXEC DOM command to delete the message again.
4	Set when the WTOR is issued with Msgtext truncated because it exceeded the maximum length.
8	Set when invalid syntax is detected. These errors are flaged by TSO/E IKJxxxxx messages or by AutoOPERATOR short error messages or both.

Examples

This section contains examples using the IMFEXEC WTO command statement. A brief discussion follows each example.

Example 1

```
/* REXX */
"IMFEXEC WTO '- "DATE() TIME() "NCP IS COMING DOWN IN 5 MINUTES'"
```

This example command sends a message to the system console with the current date and time passed through the symbolic TSO built-in functions, DATE() and TIME().

Example 2

```
/* REXX */
"IMFEXEC WTO 'SHIFT CHANGE AT 6PM' ROUTCDE(1 5 14) DESC(1)"
```

This example sends the message to specific destinations.

WTOR

This command sends a message to one or more system consoles and returns an operator reply.

Command	Parameters
WTOR	'Msgtext' [JOBID(n)] [ROUTCDE(n1 ... nn)] [DESC(n1 ... nn)] [CONSOLE|CN(n)] [NAME(x) REPLY(Variable name) [WAIT(n)] [SSID(YES|NO)]

The following table describes the parameters.

Parameter	Function	Notes
Msgtext	The text of the message to send	Maximum length is 122 characters with SSID defaulting to NO; if SSID is YES, maximum length is reduced to 121.
JOBID	Job identifier to place in SYSLOG as message issuer If the JOBID parameter is not specified, the IMFEXEC WTOR command defaults to the JES job identifier for the AutoOPERATOR subsystem.	1-8 characters alphanumeric, 1st character alpha.
ROUTCDE	Routing codes to associate with the message. Refer to the IBM publication, *Routing and Descriptor Codes*, for more information.	Refer to the IBM publication *OS/390 MVS Initialization and Tuning Reference* for information about CONSOLxx.
DESC	Descriptor codes to associate with the message	Refer to the IBM publication, *Routing and Descriptor Codes*, for more information. However, only the descriptor codes of 7 and 13 or both may be specified for a WTOR.
CONSOLE CN	Specific Console ID of console to receive the message	Numeric identifier. You can specify NAME() or CONSOLE(), but you can not specify both. If you omit the CONSOLE|CN and NAME keywords, the system uses the routing code specified on the ROUTCODE keyword on the DEFAULT statement in the CONSOLxx member of SYS1.PARMLIB. Refer to the IBM publication *OS/390 MVS Initialization and Tuning Reference* for information about CONSOLxx.

Parameter	Function	Notes
NAME	Specific name of console to receive the message	This parameter is optional. You can specify NAME() or CONSOLE(), but you cannot specify both. If you omit the CONSOLE|CN and NAME keywords, the system uses the routing code specified on the ROUTCODE keyword on the DEFAULT statement in the CONSOLxx member of SYS1.PARMLIB.
REPLY	The variable name that receives the operators' reply	1-32 alphanumeric characters.
WAIT	The number of seconds to wait for the operators' reply	If not specified, the EXEC will wait indefinitely.
SSID	Appends subsystem identifier to the end of the message	Either YES or NO.

Routing codes and descriptor codes are documented in the IBM publication, *Routing and Descriptor Codes*. However, only the descriptor codes of 7 and 13 or both may be specified for a WTOR.

Condition codes are listed in the following table.

Value	Description
0	Command was executed successfully
4	Set when the WTOR is issued with Msgtext truncated because it exceeded the maximum length or when an invalid descriptor code is received.
8	Set when invalid syntax is detected. These errors are flaged by TSO/E IKJxxxxx messages or by AutoOPERATOR short error messages or both, or when the WAIT() interval expires because no reply was received.

Example

The below example issues a WTOR. If the operator does not respond within 60 seconds or responds with an S, the CLIST branches to label STARTNET. Otherwise, execution continues sequentially.

Some common options are dependent.

```
/* REXX */
"IMFEXEC  WTOR 'NETWORK COMING UP IN 1 MINUTE, REPLY ""S"" TO STOP'",
"WAIT(60),REPLY(REP)"
IF IMFCC = 8 THEN SIGNAL STARTNET
IF REP ^= 'S' THEN SIGNAL STARTNET
```

Chapter 13. Testing and Debugging EXECs Interactively

This chapter describes testing EXECs with the AutoOPERATOR EXEC Testing and Debugging Facility.

Introduction

This section briefly introduces AutoOPERATOR EXECs and describes why and when you might use them.

Why Use AutoOPERATOR EXECs

The initial phase of implementing automation at most sites includes tasks such as message suppression or message rewording. You can simply automate these tasks by creating Rules with the AutoOPERATOR Rules Processor applications. Refer to the chapters about Rules in the *MAINVIEW AutoOPERATOR Basic Automation Guide* for more information.

As automation becomes more complex, you may find that you need more tools that are provided by a programming language. AutoOPERATOR EXECs provide an extensive programming language that comes in two formats: IBM REXX and IBM TSO CLIST. Although AutoOPERATOR has expanded the scope of automation through Rules, you may find that some automation tasks require the use of AutoOPERATOR EXECs.

With AutoOPERATOR EXECs and other facilities such as the Rules Processor applications and the ALERT Management Facility, almost every automation related problem can be solved. Information about the Rule Processor applications and the ALERT Management Facility can be found in the *MAINVIEW AutoOPERATOR Basic Automation Guide*.

What AutoOPERATOR EXECs Are

AutoOPERATOR REXX and CLIST EXEC formats contain a customized set of commands called IMFEXEC commands that address a variety of problems pertaining to data center automation.

As with every programming language, writing AutoOPERATOR EXECs introduces a learning curve and extends the development-testing-debugging cycle. Some programming languages offer debuggers to shorten this cycle. AutoOPERATOR provides the EXEC Testing Facility that allows you to interactively test your AutoOPERATOR EXECs.

What the EXEC Testing Facility Provides

The EXEC Testing Facility includes many common debugging features such as:

- Real-time execution control
- Variable read/write access
- The establishment of breakpoints (an instruction in the program which, when encountered, suspends execution and returns control to the tester)

Using the testing facility, you can review and change the state of the program or its environment during processing, or you can terminate the test altogether. You can debug EXECs in a non-standalone system so the EXEC Testing Facility also incorporates special precautions to safeguard the status of the production system.

The following sections introduce the general concepts of the EXEC Testing Facility and describe how to use the facility to debug AutoOPERATOR EXECs. A short example demonstrating different functions is also provided.

Overview

The EXEC Testing Facility provides a full-screen interactive interface you can use to debug AutoOPERATOR EXECs. During testing, the execution of the EXEC is totally under your control. At all times, it displays the EXEC with the current statement highlighted and allows you to see immediately information such as variable values and the states of OSPI sessions which would otherwise take time to obtain.

Two primary display formats are supplied: one for the less experienced user and one for the more experienced developer.

Important Note

"Minimizing EXEC Processing Time" on page 104 describes how to use the MVS Virtual Lookaside Facility to minimize the amount of CPU used by EXECs. The EXEC Testing Facility **does not allow you to test EXECs stored in VLF cache more than once per SS session**.

The first time you attempt to test an EXEC, the Testing Facility gets control of the EXEC with TSO OPEN SYSPROC and the test is run. However, if you attempt to test the EXEC a second time, EXEC will be scheduled and the Testing Facility is **bypassed**.

This occurs because once the EXEC is read into the VLF cache, the EXEC Testing Facility cannot get control over the execution of the EXEC. For more information, refer to "Minimizing EXEC Processing Time" on page 104.

What Breakpoints Are

Breakpoints are spots you can set within an EXEC that suspend execution of the program and allow you to control the EXEC.

Scenario: An EXEC gathers data in the first hundred lines of its code and VPUTs these variables to the LOCAL variable pool. The EXEC retrieves these variables, produces a report, and terminates. Suppose you find that the report seems to contain errors and you believe its because the LOCAL variables were set incorrectly.

The problem is that LOCAL variables are deleted when the EXEC terminates so you cannot verify how the variables were set. With the testing facility, you can stop the EXEC **before the report is produced**. Now, you can review the variables' settings and perhaps find that that you have to change the EXEC to produce the desired results.

This approach is preferable to creating a loop within the program that writes the contents of all variables to the AutoOPERATOR log, especially if there are a lot of variables in question.

Division of Breakpoints

Breakpoints are divided into two groups: **Unconditional** and **Conditional**.

Unconditional Breakpoints

Unconditional breakpoints cause an EXEC to always suspend execution before or after a statement.

You can set unconditional breakpoints before or after any IMFEXEC statement. This means that whenever the EXEC encounters an unconditional breakpoint, the program stops and you have control of the EXEC. The program remains suspended until you issue a command for it to resume processing.

Conditional Breakpoints

Conditional breakpoints are not associated with a particular program statement. Instead, they can be associated with the contents of one or more variables. These breakpoints may be combined using Boolean operators (such as and, or, and/or). There are many applications for this type of breakpoint.

For example, you might have a program that produces a report on all DASD devices in the system. An inner loop obtains information for one device at a time and then calls a subroutine to print a status line. Suppose the status for one device, for example unit address AFC, always seems to be incorrect.

Setting an unconditional breakpoint at the beginning of the subroutine is not helpful because the breakpoint is met for every device and your data center may have hundreds of DASD devices.

The better solution is to:

- Locate the variable containing the current unit address for which reporting is to be done
- Establish a conditional breakpoint that suspends program execution when this variable contains the value AFC
- Restart the program

Setting a conditional breakpoint causes the program to stop whenever processing for this device is about to begin and you can review surrounding code.

How to Use Variables

Special precautions have been taken so that an EXEC being tested has VGET access but no VPUT/VDEL access to PROFILE and SHARED variables. This is because the PROFILE and SHARED variables may be in use and shared by other EXECs in production.

Therefore, an EXEC being tested possesses two additional variable pools: the PROFILE TEST and SHARED TEST pool. These two pools are logically concatenated ahead of the live pools so that VGET requests may be satisfied from the actual SHARED and PROFILE pools while VPUT and VDEL requests are always directed to the test pools.

However, these new pools now create a problem in the following scenario.

Scenario: Assume variable ABC exists in the SHARED pool prior to the debugging of an EXEC. When the EXEC initially VGETS its value, the test pool is searched first. Because variable ABC is not in the test pool, it is retrieved from the SHARED pool. Subsequently, the EXEC issues a VDEL ABC SHARED command. This removes the variable from the test pool, but does not access the live pool. If another VGET ABC SHARED statement follows in the EXEC, the variable would be retrieved from the SHARED pool again which would be the incorrect thing to do.

To protect against this scenario, the EXEC Testing Facility maintains a list of all DELETE requests for the variables in the SHARED and PROFILE pool and checks this list before attempting to retrieve a value from either of these pools. Note while this eliminates the dilemma described above it may also be confusing because another EXEC which is not being tested might access the variable.

Important Note

A variable that has been deleted from the SHARED or PROFILE pool using explicit or pattern matching means that it still exists in those pools but all attempts to access it with VGET or VPUT commands are treated as if it had actually been removed.

Using the EXEC Testing Facility with OSPI EXECs

The Open Systems Procedural Interface allows EXECs to interact with applications that usually exchange information only with physical terminals. The EXEC Testing Facility allows you to determine the state of OSPI sessions which can help you determine why your OSPI EXEC is not working.

Scenario: Assume an OSPI EXEC logs on to a TSO user ID. At the TSO READY prompt, you enter `PDF` to invoke ISPF/PDF. At some point, the EXEC begins issuing error messages indicating that you attempted to enter data into a protected field. You can invoke the EXEC in the testing facility and stop the EXEC with an unconditional breakpoint set immediately before the statement causing the error.

With the EXEC interrupted, you can display and examine the current buffer image of the EXEC. You may find that the amount of broadcast messages received during logon exceeded one screenful and if the EXEC code does not take this possibility into consideration then no input fields existed to enter the PDF command.

How to Use the IMFEXEC BKPT Statement

EXECs containing large amounts of logic code may be hard to debug since breakpoints can be set only at IMFEXEC statements. If an EXEC does not include any of these commands, you have control of the EXEC only upon entry. To solve this problem, AutoOPERATOR includes the IMFEXEC BKPT statement.

The IMFEXEC BKPT statement works only when the EXEC is being tested with the EXEC Testing Facility. When the EXEC is scheduled any other time, the statement produces no effect and does not perform any function. IMFEXEC BKPT also does not change the current value of IMFCC.

For testing purposes, you can set breakpoints with IMFEXEC BKPT in the EXEC where normally no IMFEXEC statement would be found. The IMFEXEC BKPT statement imposes minimal overhead and it does not impact the performance of the EXEC so when the EXEC is moved into production, these statements, like comments, do not need to be removed.

How to Trace the Execution of the EXEC

The EXEC Testing Facility produces a history of the Another feature of the EXEC Testing Facility is the ability to review the execution history of the tested EXEC. Sometimes, an EXEC may branch to a particular statement and it is not apparent how it got there. The debugging trace contains, among other things, all recently executed IMFEXEC statements. This history provides a good indication about the logic path the program has taken. The trace information, similar to system traces, is maintained in a wrap-around buffer and wraps after collecting 379 lines of data. This means the most recent information is always available and the trace buffer never runs out of storage.

What to Set Up Before Using the EXEC Testing Facility

Before any EXECs can be tested, a minimum of setup work is required. These EXECs may require that the IMFxxxx variables contain certain values or that parameters being passed to the EXEC are present.

This setup work is relatively straightforward but may be time-consuming. In many cases, an EXEC requires more than one iteration of the coding-debugging cycle, during which the time spent for test setup can become significant.

To speed up this process, a SAVE feature has been built into the variable related application. SAVE allows all TSO variables to be saved with their current values and to be retrieved at a later point in time while debugging either the same or any other EXEC. The data is stored in the AutoOPERATOR subsystem (SS) so that it can easily be reused not only by the current user but also by any other user. Predefined customized patterns by EXEC type (ALERT-initiated EXEC, Rule-initiated EXEC and so on) may be stored. Later, they can serve as patterns that require minimal modification. The storage mechanism is self-reorganizing and geared towards multi-user access.

Accessing the EXEC Testing Facility

Access the EXEC Testing Facility through the EXEC Management Facility. Figure 20 shows an example.

```
BMC Software ------------------ EXEC Management ----------------- AutoOPERATOR
COMMAND ===>                                                    TGT ===> SYSC
INTERVAL ==> 1                                                  DATE --- 01/01/29
STATUS   --- INPUT              Scroll  right/left             TIME --- 07:29:58
Primary command: Sort
         EXECs defined   159     Scheduled     2            Enabled   159
 High Priority running     0        Queued     0                        1
 Norm Priority running     0        Queued     0                        7

 PRESS EXPAND TO VIEW EXEC ACTIVITY
 (B)ROWSE, (E)NABLE, (D)ISABLE, (S)ELECT EXEC, (T)EST EXEC

LC  NAME     STATUS   GROUP    FUNCTION CODE AUTHOR   DESCRIPTION
 __ ________ _        ________ ________ __   ________ ______________________________
_   PAEXP01  ENABLED                        CIM4
_   PAEXP02  ENABLED                        CIM4
_   PAGPNL   ENABLED                        CIM4
_   PALIST   ENABLED                        CIM4X
T   PARSE    ENABLED  SYSTEMS  UTIL         ERNST    ENQ CHECKER FOR MULTISYST
_   PATTERN  ENABLED                        RAE2
_   POST     ENABLED                        RAE1
_   PRGAAO   ENABLED                        RAE1
_   PROV1    ENABLED                        RAE1
```

Figure 20. EXEC Management Application Panel

Once the EXEC Management panel is displayed, to test an EXEC:

1. Locate the EXEC name in the member list.
2. Place a `T` (for Test EXEC) in the `LC` column.

   In Figure 20, an EXEC named PARSE is selected to be tested.
3. Press the ENTER key.

The EXEC is loaded and the EXEC Test panel is automatically displayed (Figure 21 on page 418).

```
BMC Software ---------------------- EXEC Test --------------------------------
COMMAND ===>
EXEC === PARSE     ID === 3                                   DATE --- 01/01/01/29
Options: B - Browse EXEC output        V - Variable access   TIME --- 07:59:48
         C - Conditional Breakpoints   O - OSPI session display
Primary Commands:
    STEP - Single step execution       CONTinue - Execute, stop at breakpoints
    RUN  - Execute without stopping    CANcel   - Terminate execution
    L    - Locate by line number       Find     - Find string
    OFF  - Remove all breakpoints      CMDSHOW ON/OFF -Show/remove this display
    SKIP - Skip current statement      FORCE    - Force terminate EXEC
    VARON/VAROFF | EXPAND - Variable substitution on current line
Line Commands --- Break (A)fter, Break (B)efore, (O)ff
         ************************************************************
       B->PROC 2 P1 DSNAME                                             00001
          /************************************************************ 00002
          /*                                                           00003
          /* DOC GROUP(SYSTEMS) AUTHOR(ERNST) FUNC(UTIL)               00004
          /* DOC DESC(ENQ CHECKER FOR MULTISYSTEM)                     00005
          /*                                                           00006
          /************************************************************ 00007
          GLOBAL NROFTARGETS                                           00008
          SET EOF=FALSE                                                00009
          SET NROFTARGETS=0                                            00010
```

Figure 21. EXEC Test Control Panel

The EXEC Test Control panel shows a scrollable listing of the EXEC at the bottom of the panel. The upper portion of the panel contains information about the available commands. Whenever an EXEC is tested, the debugger suspends execution before the first statement.

The current position of the EXEC during testing is indicated by three factors:

- The line is highlighted
- An arrow to the left of it points at it
- The character `B` (for before) or `A` (for after) prefixes it

The field near the top of the panel, labeled `EXEC`, contains the name of the currently tested member. When working with nested EXECs (EXECs that call other EXECs as subroutines using the IMFEXEC SELECT WAIT(YES) command), the debugger always reflects the name of the EXEC currently executing. Therefore, the value of the `EXEC` field can change during the debugging session and may be used to establish a point of reference for the level of processing.

If you are debugging an EXEC (EXECA) that calls another EXEC (EXECB) with IMFEXEC SELECT WAIT(NO), then EXECB will execute and the debugging application has no control over EXECB and EXECB cannot be debugged.

The field `ID` shows the identification assigned to the current thread, so it remains constant throughout the debugging session. This field also reflects the number of EXECs executed since the last AutoOPERATOR SS restart (this number does not include nested EXECs).

Since a large portion of the panel is taken up by command information that is not required by the more advanced user, the bottom portion of the panel can be expanded to cover the upper portion of the panel.

To expand the view, issue the `CMDSHOW OFF` primary command. The panel will look like Figure 22:

```
BMC Software ---------------------- EXEC Test ------------------- AutoOPERATOR
COMMAND ===>
EXEC === PARSE     ID === 3
         ************************************************************
       B->PROC 2 P1 DSNAME                                            00001
          /************************************************************ 00002
          /*                                                          00003
          /* DOC GROUP(SYSTEMS) AUTHOR(ERNST) FUNC(UTIL)              00004
          /* DOC DESC(ENQ CHECKER FOR MULTISYSTEM)                    00005
          /*                                                          00006
          /************************************************************ 00007
          GLOBAL NROFTARGETS                                          00008
          SET EOF=FALSE                                               00009
          SET NROFTARGETS=0                                           00010
          ERROR DO                                                    00011
            IF &LASTCC=400 THEN DO                                    00012
              SET EOF=TRUE                                            00013
              CLOSFILE MYJNT                                          00014
              RETURN                                                  00015
            END                                                       00016
            ELSE DO                                                   00017
              EXIT                                                    00018
            END                                                       00019
          END                                                         00020
```

Figure 22. EXEC Test Control Panel—Advanced Format

Commands

The following application commands can be issued on the COMMAND line:

Command	Function
B	Browses the EXEC trace wraparound buffer.
C	Sets and resets conditional breakpoints.
V	Displays, deletes, and changes variable contents.
O	Displays OSPI buffer images.

In addition to these commands, the following primary commands can be issued at the COMMAND line:

Command	Function
CANCEL	Terminates execution of the EXEC and returns to the EXEC Management panel. This command is successful only if the EXEC has returned control to you by using a breakpoint (see “How to Use the IMFEXEC BKPT Statement” on page 416 for more information).

CMDSHOW Uses the following parameters to control the presentation format of the EXEC Test Control panel:

ON Turns command help on
OFF Turns command help off

CONTINUE Resumes EXEC processing until the next breakpoint is encountered.

EXPAND Displays the interpreted buffer image of the current line with all functions and variables substituted. A separate panel is used to accommodate continuation lines.

Find Attempts to locate the argument in the current EXECs source and repositions the display accordingly.

FORCE Terminates execution of the EXEC. This command should be used as a last resort. It functions similar to the .CANCEL BBI command.

GO Is an alias for CONTINUE.

Locate Uses the argument so that the relative line with the given line number is displayed at the top.

OFF Removes all conditional or unconditional breakpoints set for the currently displayed EXEC.

RUN A combination of OFF and CONTINUE. All breakpoints are removed and EXEC processing resumes.

SKIP When positioned before an IMFEXEC statement, this command skips processing of that specific IMFEXEC statement and resumes EXEC processing.

STEP Resumes execution of the EXEC until the next IMFEXEC statement is encountered and the STEP command returns control to you.

VARON Turns current line interpretation on. The current line is shown in interpreted instead of source line format.

VAROFF Turns current line interpretation off. The current line is shown in source line instead of interpreted format. This is the initial setting.

Using Line Commands

The following line commands can be entered in the input field (shown as an underscore character to the left of every IMFEXEC statement in the source):

Command	Function
A	Establishes a breakpoint after the selected statement. Execution control is returned to you immediately after processing this statement. An `A` to the left of the statement serves as a reminder that this type of breakpoint has been set.
B	Establishes a breakpoint before the selected statement. Execution control is returned to you immediately before processing this statement. Use this form of breakpoint if you want to use the SKIP primary command. A `B` to the left of the statement serves as a reminder that this type of breakpoint has been set.
O	Removes all breakpoints from the indicated line.

Displaying Interpreted Source Statements

This command toggles the display of the current source line to an interpreted format. This means that the following have been completed on this line:

- All TSO/REXX functions have already been executed
- All variables have been evaluated and replaced by their values

This is most easily understood by looking at the following two panels. Figure 23 shows the source code exactly as it is found in the member of the SYSPROC concatenation. The current statement contains the variable name `&NROFTARGETS`.

```
BMC Software ---------------------- EXEC Test -------------------- AutoOPERATOR
COMMAND ===>
EXEC === PARSE     ID === 3                                  DATE --- 01/01/29
Options: B - Browse EXEC output        V - Variable access   TIME --- 14:50:44
         C - Conditional Breakpoints   O - OSPI session display
Primary Commands:
    STEP - Single step execution       CONTinue - Execute, stop at breakpoints
    RUN  - Execute without stopping    CANcel   - Terminate execution
    L    - Locate by line number       Find     - Find string
    OFF  - Remove all breakpoints      CMDSHOW ON/OFF -Show/remove this display
    SKIP - Skip current statement      FORCE    - Force terminate EXEC
    VARON/VAROFF | EXPAND - Variable substitution on current line
Line Commands --- Break (A)fter, Break (B)efore, (O)ff
         ************************************************************
              SET I = &I + 1                                      00052
            END                                                       00053
            IF &TYPE=MVS THEN DO                                      00054
              SET NROFTARGETS=&NROFTARGETS+1                          00055
              SET MTARGET&NROFTARGETS = &TARGET                       00056
_   B  B->    IMFEXEC VPUT MTARGET&NROFTARGETS LOCAL                  00057
            END                                                       00058
          END                                                         00059
         ************************** END OF DATA ***************************
```

Figure 23. EXEC Test Panel with the VAROFF Option

Figure 24 shows the current line after interpretation with excessive blanks and possible comments removed. It also substitutes the value of the variable `&NROFTARGETS` for its name.

Note that not only variables are substituted but also all other references are resolved. The line will always only show literals instead of functions, variables, and so on.

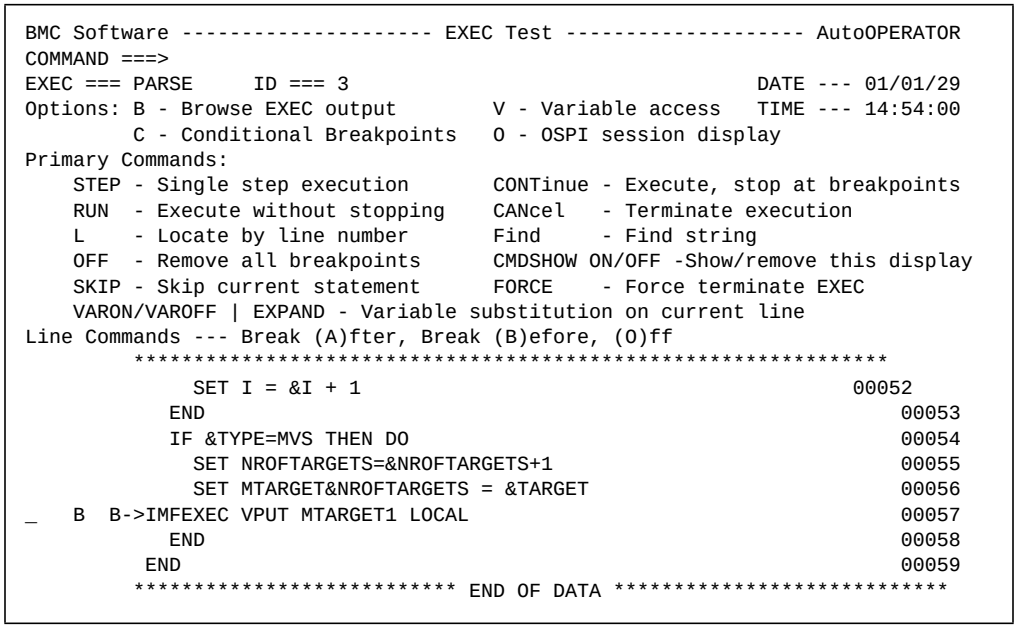


Figure 24. EXEC Test Panel with the VARON Option

Tracing Interpreted Source Statements

The EXEC Trace panel shows the most recently executed IMFEXEC statements, sorted in ascending order, by execution time. The history of the EXEC's processing can sometimes help in explaining why a certain program branch was taken. The `EXEC` and `ID` show the same information as in the EXEC Testing Control panel.

Note: The information in the EXEC Test - Trace panel is not available after the tested EXEC thread ends.

The trace information of an EXEC that is executing under the EXEC Testing Facility can be browsed (as shown in Figure 25) by entering the primary command `B` on the EXEC Testing Control panel.

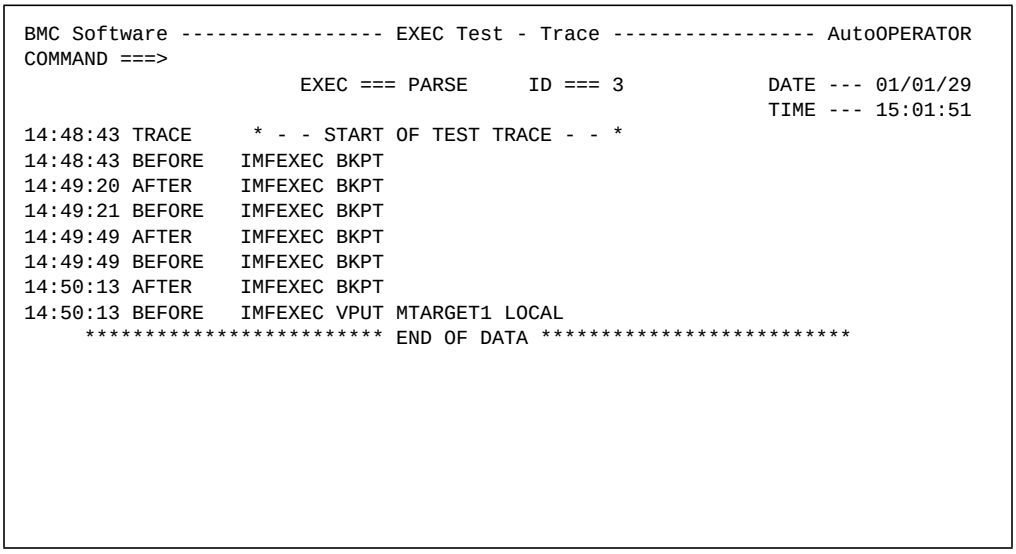


Figure 25. EXEC Trace Panel

Setting Conditional Breakpoints

The Conditional Breakpoints panel provides the capability to set and reset conditions that, when met, cause EXEC execution to be suspended and control returned to you (refer to "BKPT" on page 259 for more information about conditional breakpoints).

Note: The specified conditions are checked whenever an IMFEXEC statement is encountered. If a condition is met but no IMFEXEC statement encountered, **the breakpoint does not trigger.**

The `C` primary command issued from the EXEC Test panel invokes the Conditional Breakpoint Control panel as shown in Figure 26.

```
BMC SOftware ---------------- Conditional Breakpoints ------------ AutoOPERATOR
COMMAND ===>

Exec Name === PARSE              Exec ID == 3

 Variable-name                    Op  Value                                Pool

 NROFTARGETS_____________________ GT  1____________________________________ TSO_
 MTARGET1________________________ =_  SYSC___________________________________ LOCL
 _____________________________________ __  _____________________________________ ____
 _____________________________________ __  _____________________________________ ____
 _____________________________________ __  _____________________________________ ____
 _____________________________________ __  _____________________________________ ____
 _____________________________________ __  _____________________________________ ____
 _____________________________________ __  _____________________________________ ____
 _____________________________________ __  _____________________________________ ____
 _____________________________________ __  _____________________________________ ____
 _____________________________________ __  _____________________________________ ____
 _____________________________________ __  _____________________________________ ____
 _____________________________________ __  _____________________________________ ____
```

Figure 26. Conditional Breakpoint Control Panel

The `EXEC` and `ID` fields carry the same contents as in the EXEC Testing Control panel. You can specify up to 13 conditions that must be met for the EXEC to halt:

1. Specify the name of the variable whose contents are to be examined in one of the fields of the column labeled `Variable-Name.`
2. Enter an operator in the column labeled `OP`.
3. Enter a literal against which the comparison will be made in the column labeled `Value`.
4. Specify the pool in which the variable to be checked resides in the `Pool` column.

Following a list of valid operators:

Operator	Function
=	Equal to (or matches with literal pattern)
EQ	Equal to (or matches with literal pattern)
¬=	Not equal to
<>	Not equal to
NE	Not equal to
<	Less than
LT	Less than
>	Greater than

GT	Greater than
>=	Greater than or equal to
GE	Greater than or equal to
<=	Less than or equal to
LE	Less than or equal to

The POOL column accepts one of the following literals:

Value	**Interpretation**
TSO	Standard REXX or CLIST variable
LOCL	LOCAL AutoOPERATOR pool
SHAR	SHARED AutoOPERATOR pool
PROF	PROFILE AutoOPERATOR pool
SHRT	SHARED TEST AutoOPERATOR pool (refer to “Introduction” on page 411)
PRFT	PROFILE TEST AutoOPERATOR pool (refer to “Introduction” on page 411)

The following rules apply for comparisons:

- If the operator indicates an equal to operation and either the literal or the variable content contains non-numeric characters, a pattern matching operation is performed.

  For example, if the variable ABC contains the character string `'BAB053 BAC635 BAB059'` and the comparison operator is an equal sign and the literal specifies `'*BAC635*'`, the condition is considered met.

- If both the literal and the variable contain only numeric characters (possibly prefixed by a sign), a numeric comparison is performed which may render different results than a character-only comparison.

  For example, if the variable contains a value of '0000' and the literal specifies '0', the condition will be considered met.

  All conditions specified on this panel must be met for the EXEC to return control to you, that is, an implicit AND operator is assumed between all specifications.

Displaying Variables

This panel displays a scrollable listing of all TSO variables of the currently executing EXEC, accompanied by all variables found in the AutoOPERATOR LOCAL pool for this EXEC. In addition, all variables contained in the PROFILE, SHARED, PROFILE TEST, and SHARED TEST pool may be shown.

Issuing the primary command V in the COMMAND field of the EXEC Testing Control panel invokes the panel shown in Figure 27.

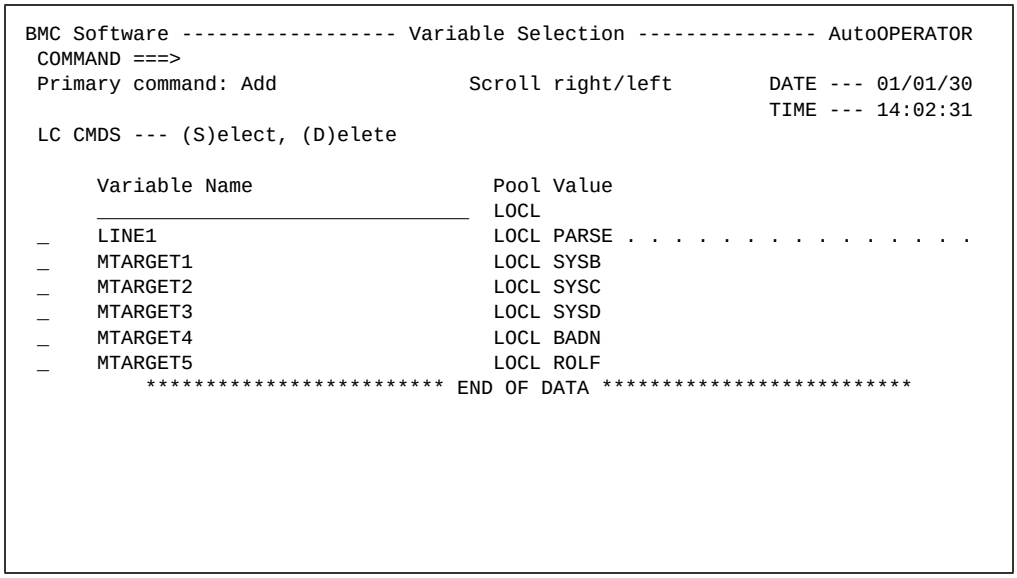


Figure 27. Variable Selection Panel

Two input fields allow you to narrow the focus down to the variables of interest. You can specify a pattern in the column labeled `VARIABLE NAME` that all variables to be displayed must match; for example, entering IMF* (or IMF for short) displays only all variables beginning with these 3 characters.

If you enter a pattern that ends with a dash (-), you must enter an asterisk after the dash. For example:

```
IMF-*
```

The input field in the column titled POOL accepts either a full or pattern specification for one of the following values:

Value	Interpretation
TSO	Standard REXX or CLIST variable
LOCL	LOCAL AutoOPERATOR pool
SHAR	SHARED AutoOPERATOR pool
PROF	PROFILE AutoOPERATOR pool
SHRT	SHARED TEST AutoOPERATOR pool (refer to “Introduction” on page 411)
PRFT	PROFILE TEST AutoOPERATOR pool (refer to “Introduction” on page 411)

For example, specifying P* displays all variables currently contained in the PROFILE and PROFILE TEST pools. This display can be scrolled right and left to show the full value of a variable.

The following line commands can be entered in the input fields in the column labeled LC:

Command	Interpretation
S	Select this variable for update or display
D	Delete this variable

The ADD primary command may be used to add additional variables to a particular pool. PROFILE and SHARED variables may be displayed but not deleted or modified. Refer to “Overview” on page 413 for an explanation for this restriction.

Creating and Modifying Variables

The Variable Add/Update panel shows the complete contents of the selected variable. If the variables resides in the SHARED or PROFILE pool, the contents can be examined but **not changed**. Otherwise, variable name, pool, and contents are input fields and may be overtyped with new values.

Issuing the ADD primary command or the S line command on the Variable Selection panel invokes the panel shown in Figure 28.

```
BMC Software ----------------- Variable Add/Update ---------------- AutoOPERATOR
COMMAND ===>
Primary command: HEX ON/OFF                                  DATE --- 01/01/30
                                                             TIME --- 14:05:45
Variable Name  ===> MTARGET1
Pool           ===> LOCL
Value (Enter Below):
SYSB









To update variable, press END  To cancel changes, enter CANCEL
```

Figure 28. Variable Add/Update Panel

The `Pool` input field can contain or accept the following values:

Value	Interpretation
TSO	Standard REXX or CLIST variable
LOCL	LOCAL AutoOPERATOR pool
SHAR	SHARED AutoOPERATOR pool (may not be entered)
PROF	PROFILE AutoOPERATOR pool (may not be entered)
SHRT	SHARED TEST AutoOPERATOR pool (refer to “Overview” on page 413)
PRFT	PROFILE TEST AutoOPERATOR pool (refer to “Overview” on page 413)

Keep in mind that you may not make modifications to the PROFILE or SHARED pools.

The primary command HEX with the parameters ON and OFF toggles the display to a hexadecimal representation and back as shown in Figure 29.

```
BMC Software ----------------- Variable Add/Update ---------------- AutoOPERATOR
COMMAND ===>
Primary command: HEX ON/OFF                                    DATE --- 01/01/30
                                                               TIME --- 14:06:34
Variable Name  ===> MTARGET1
Pool           ===> LOCL
Value (Enter Below):
 SYSB
 EEEC
 2822
  ------------------------------------------------------------------------------



  ------------------------------------------------------------------------------



  --------------------



To update variable, press END  To cancel changes, enter CANCEL
```

Figure 29. Variable HEX Display

Updates may be performed in either presentation format.

Testing OSPI Sessions

This panel contains a list of all OSPI Sessions established during the current debugging session that have not been explicitly terminated using the IMFEXEC LOGOFF command.

This panel is invoked by the O primary command of the EXEC Testing Control panel as shown in Figure 30:

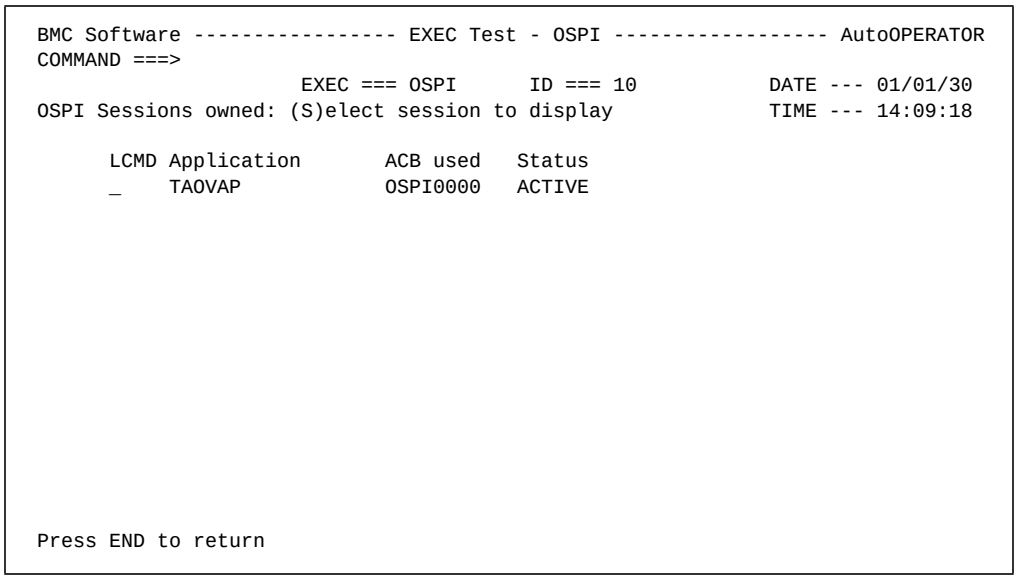


Figure 30. OSPI Session Panel

Following is a description of the columns on this panel:

Title	Description
Application	VTAM ACB name of the application that a session was established to.
ACB used	The VTAM ACB either explicitly specified on the IMFEXEC LOGON statement using the REQACB keyword or implicitly picked from the pool assigned to OSPI.
Status	One of the following:
	ACTIVE The session is currently active. Commands may be entered against it.
	INACTIVE The session has been terminated either due to a failure or because an UNBIND was sent from the session partner. Any DISCONNECTed sessions are not shown.

You can see the current buffer image of the session by placing a S in the `LC` column and pressing ENTER. The buffer image is displayed until you press ENTER again.

Chapter 14. Using the AutoOPERATOR-Supplied Utility EXECs

BMC Software provides a set of utility EXECs located in BBPARM member BBPROC. These EXECs perform functions or subroutines that sometimes are called by various other EXECs (for example, as an EXEC-initiated EXEC). These EXECs also can be invoked by an operator (for example, as an user-initiated EXEC).

The return codes returned by the EXECs and their meanings are listed with the discussion of each EXEC. To test a return code, use the variable IMFRC.

"How to Resolve Compound SYSPROG Variables" on page 433 describes calling conventions and requirements.

Distributed Utility EXECs

The following lists the distributed utility EXECs.

EXEC Name	Description
@STATASK	Command interface to start tasks
CANEXEC	Cancels DELVARS on user specification
DELVARS	Deletes variables from a pool
MUT001C	Issues $E, $P, and $C JES2 commands
SUBMIT	Determines which SS will handle job submission
SUBMITOR	Submits jobs on the target SS
@TIMER	Interface EXEC to timer queues; this drives EXECs/events
JES2DI	Interface EXEC to JES2's $DI command
JES2DQ	Interface EXEC to JES2's $DQ command
CNVSECS	Convert time HH:MM:SS format to seconds
CNVTIME	Convert seconds to HH:MM:SS format

SYSPROG Utility EXECs

The following lists the utility EXECs that allow you to interface with SYSPROG services. The following naming convention is used for all the SYSPROG utility EXECs:

Rxxx

where xxx is the first three characters of the SYSPROG service.

EXEC Name	Description
RASM	Interface to the SYSPROG ASM service
RCPU	Interface to the SYSPROG CPU service
RCSS	Interface to the SYSPROG CSSUM service
RIO	Interface to the SYSPROG IO service
RMDE	Interface to the SYSPROG MDEV service
RMON	Interface to the SYSPROG MON service
RMPA	Interface to the SYSPROG MPA service
RMTP	Interface to the SYSPROG MTP service
RPAG	Interface to the SYSPROG PAGING service
RPRO	Interface to the SYSPROG PRO service
RREP	Interface to the SYSPROG REP service
RREPRX	Interface to the SYSPROG REP service
RRES	Interface to the SYSPROG RES service
RRSM	Interface to the SYSPROG RSM service
RSTA	Interface to the SYSPROG STA service
RSYS	Interface to the SYSPROG SYS service
RTPI	Interface to the SYSPROG TPIO service
RTSU	Interface to the SYSPROG TSU service

How to Resolve Compound SYSPROG Variables

The SYSPROG utility EXECs are called by other EXECs with the WAIT(YES) parameter. These EXECs produce LOCAL variables that are available to their callers upon return; the variables are prefixed with the name of the EXEC. The LOCAL variables can be accessed with the IMFEXEC VGET command by naming the specific variable or in a do-loop if more than one variable of the same type is required.

Caution:
Compound variables in the format of RREPI(xxx) need to be resolved prior to their usage in any function; otherwise, the results are unpredictable.

An example using a SYSPROG utility EXEC is shown in Figure 31.

```
/*REXX*/
"IMFEXEC SELECT EXEC(RREP) WAIT(YES)"   <=== CALL REPLIES EXEC
"IMFEXEC VGET (RREPROL1) LOCAL"         <=== NUMBER OF OUTPUT LINES
                                             RETURNED IN LOCAL POOL
N = 1
"IMFEXEC MSG 'LIST OF OUTSTANDING REPLIES'"
DO WHILE N <= RREPROL1
  "IMFEXEC VGET (RREPI"N" RREP"N") LOCAL"   <=== LOCAL VARIABLE
  "IMFEXEC MSG .REPLYID="VALUE('RREPI'N) "MESSAGE ="VALUE('RREP1'N)
   N = N +1
END
```

Figure 31. Example of SYSPROG Utility Usage

@STATASK: Start Tasks

The EXEC @STATASK starts a specific task. It ensures that the task is not already active before issuing the START command.

The following statement shows the format of the call from an EXEC:

```
IMFEXEC SELECT EXEC(@STATASK TASKNAME ID PARM)
```

Table 11 lists the @STATASK parameters.

Table 11. @STATASK Parameters

Parameter	Required?	Description
TASKNAME	Yes	Name of the task to start; for example: `RESOLVE`
ID	No	ID name of the task; for example: `RESOLVE.R` where R is the ID
PARAMETER	No	Any parameters for the task. If required, include the PARM= keyword; for example: `PARM=SOFTIPL`

Following is a list of return codes from @STATASK:

Return Code	Description
0	Request successfully completed
8	Invalid parameters specified
12	Task is already active
16	Task failed to start

CANEXEC: Cancel Delvars

This EXEC cancels the DELVARS EXEC. When the ALERT produced by DELVARS is selected, the ALERT application schedules the CANEXEC.

The CANEXEC is an internal utility and there are no external user call interfaces.

Following is a list of return codes from CANEXEC:

Return Code	Description
0	Request successfully completed
12	EXEC ID to cancel not passed

DELVARS: Delete Variables

This EXEC is invoked either by other EXECs or by the operator to delete all or selected variables from a specified pool.

The following statement shows the format of the call from EXEC:

```
IMFEXEC SELECT EXEC(DELVARS PARM POOL)
```

Use the following format for a console-initiated request:

```
%DELVARS PARM POOL
```

If the parameter POOL is not specified, POOL defaults to SHARED pool.

Table 12 lists the DELVARS parameters.

Table 12. DELVARS Parameters

Parameter	Required?	Description
PARAMETER	Yes	Name of the variable to delete; to delete all variables from a pool, specify an asterisk (*) or ALL
POOL	No	If specified, it must be SHARED or PROFILE; default is SHARED
TYPE	[SHORT|LONG]	Specifies whether the EXEC is to delete long or short variables

Following is a list of return codes from DELVARS:

Return Code	Description
0	Request successfully completed
8	Variable does not exist
12	Specified POOL or TYPE is incorrect

MUT001C: Issue $E, $P, and $C Commands

This EXEC is invoked by the JESDOWN EXEC to reset and drain a specified line, remote, or printer. It also can reset and cancel a specified job.

The following statement shows the format of the call from EXEC:

```
IMFEXEC SELECT EXEC(MUT001C RMLNPRJB RLPJ)
```

Table 13 lists the MUT001C parameters.

Table 13. MUT001C Parameters

Parameter	Required?	Description
RMLNPRJB	Yes	JES2 REMOTE, LINE, PRINTER, or JOBNAME to issue $E/$P/$C commands against
RLPJ	Yes	A one character literal indicating that the first value is a R(emote), L(ine), P(rinter) or J(job)

There are no return codes from MUT001C.

SUBMIT: Find Subsystem Handling Job Submissions

This EXEC is invoked by any EXEC or TS user who needs to submit a batch job. The SUBMIT EXEC VGETs the value of the SHARED pool variable SUBMITSS. Then it calls the SUBMITOR EXEC to submit the job on the target SS whose identification was VPUT at initialization.

The following statement shows the format of the call from EXEC:

```
IMFEXEC SELECT EXEC(SUBMIT JOBNAME)
```

Use the following format for a TS-initiated request:

```
%SUBMIT JOBNAME
```

Table 14 lists the SUBMIT parameters.

Table 14. SUBMIT Parameters

Parameter	Required?	Description
JOBNAME	Yes	Name of the job to submit

Following is a list of return codes from SUBMIT:

Return Code	Description
0	Call to SUBMITOR successful
8	Cannot VGET SUBMITSS variable
12	No jobname passed

SUBMITOR: Submit Jobs on the Target Subsystem

This EXEC is invoked by the SUBMIT EXEC or other EXEC to submit a batch job.

The following statement shows the format of the call from EXEC:

```
IMFEXEC SELECT EXEC(SUBMITOR JOBNAME)
```

Table 15 lists the SUBMITOR parameters.

Table 15. SUBMITOR Parameters

Parameter	Required?	Description
JOBNAME	Yes	Name of the job to submit

Following is a list of return codes from SUBMITOR:

Return Code	Description
0	Job was successfully submitted
12	No jobname passed

RASM: Auxiliary Storage Manager Information

Use the RASM EXEC to determine the last IPL type, the address of the ASMVT, the total number of slots, the number and percentage of available slots, the largest user of slots, and the percentage owned along with information on all the paging volumes.

The following statement shows the format of the call from EXEC:

```
IMFEXEC SELECT EXEC(RASM OPTS) WAIT(YES)
```

Table 16 lists the RASM parameters.

Table 16. RASM Parameters

Parameter	Required?	Description
OPTS	No	If specified, must be MAP
SORTFLD	No	Name, VIO NVIO (Corresponds to SYSPROG input parameters)

Following is a list of return codes from RASM:

Return Code	Description
8	Security failure
16	Parameter neither null nor map

Table 17 lists the variables returned by RASM in the LOCAL POOL.

Table 17. Variables Returned by RASM in the LOCAL POOL

Variable Name	Type	Description
RASMB(XXX)	Data	Burst
Note: Where XXX corresponds to the output line number		
RASMF(XXX)	Data	Free slots
RASMIPLT	Data	IPL type
RASMJ(XXX)	Data	Jobname
RASML(XXX)	Data	Label of Volume
RASMN(XXX)	Data	Non-VIO slots
RASMP(XXX)	Data	Percentage free
RASMROL1	Control	Number of output lines
RASMROL2	Control	Start line number of page data set information
RASMROL3	Control	Start line number of map information
RASMS(XXX)	Data	Size of data set in slots
RASMSLAV	Data	Available slots
RASMSLPC	Data	Percentage available
RASMSUSR	Data	Largest slot user
RASMT(XXX)	Data	Type of page data set
RASMTLSL	Data	Total slots
RASMU(XXX)	Data	Unit address of page data set
RASMUSPC	Data	Percentage largest user is holding
RASMV(XXX)	Data	VIO slots
RASMVTA	Data	Address of vector table

RCPU: CPU Usage Information

Use the RCPU EXEC to determine the top 10 CPU users, overall MVS overhead, total batch usage, total TSO usage, and overall CPU busy.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RCPU OPTS) WAIT(YES)
```

Table 18 lists the RCPU parameters.

Table 18. RCPU Parameters

Parameter	Required?	Description
OPTS	No	Number of seconds to monitor CPU usage; if not specified, the default is 10 seconds

Following is a list of return codes from RCPU:

Return Code **Description**

8 Security failure

Table 19 lists the variables returned by RCPU in the LOCAL POOL for non-PR/SM systems.

Table 19. Variables Returned by RCPU in the LOCAL POOL for Non-PR/SM Systems

Variable Name	Type	Description
RCPUB(XX)	Data	CPU percent busy
RCPUBATP	Data	Percentage of CPU used by batch
RCPUBATT	Data	Seconds of CPU used by batch
RCPUC(XX)	Data	CPU number
RCPUD(XX)	Data	Dispatching priority
RCPUMSVO	Data	Seconds of CPU in MVS overhead
RCPUMVSP	Data	Percentage of CPU in MVS overhead
RCPUN(XX)	Data	Name of job
RCPUP(XX)	Data	Priority
RCPUROL1	Control	Total number of output lines
RCPUROL2	Control	Number of job related output variable groups
RCPUROL3	Control	Number of CPU related output groups
RCPUS(XX)	Data	Seconds of CPU used
RCPUT(XX)	Data	Type of JOB (STC BAT TSU)
RCPUTSOP	Data	Percentage of CPU used by TSO

Table 19. Variables Returned by RCPU in the LOCAL POOL for Non-PR/SM Systems (Continued)

Variable Name	Type	Description
RCPUTSOT	Data	Seconds of CPU used by TSO
RCPUU(XX)	Data	Percentage of CPU

Table 20 lists the variables returned by RCPU in the LOCAL POOL for PR/SM systems.

Table 20. Variables Returned by RCPU in the LOCAL POOL for PR/SM Systems

Variable Name	Type	Description
RCPUT(XX)	Data	Type of address space (STC JOB or TSO)
RCPUN(XX)	Data	Address space name
RCPUU(XX)	Data	Percentage of CPU used
RCPUP(XX)	Data	Dispatching priority in hexadecimal
RCPUD(XX)	Data	Dispatching priority in decimal
RCPUBATP	Data	Percentage of CPU used by batch address space
RCPUSTC	Data	Percentage of CPU used by started task
RCPUTSOP	Data	Percentage of CPU used by TSO users
RCPUTTAL	Data	Total: always 100%
RCPUROL1	Control	Total number of output lines
RCPUROL2	Control	Number of job related output variable groups
RCPUBUSY	Data	Percent of time total complex was processing
RCPUWAIT	Data	Percent of time total complex was waiting
RCPUOVHD	Data	Percent of time spent on plex overhead
RCPURCVD	Data	Percent of time this partition received
RCPURLTV	Data	CPU percent which is this partition'S relative share
RCPUOTHR	Data	Percent CPU used by address spaces not listed
RCPUTOTL	Data	Total CPU percent this partition used of its relative share

RCSS: Common Storage Usage Information

Use the RCSS EXEC to determine the virtual storage usage on an address space basis.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RCSS) WAIT(YES)
```

There are no RCSS input parameters.

Following is a list of return codes from RCSS:

Return Code	Description
8	Security failure
16	COMMON STORAGE MONITOR not enabled

Table 21 lists the variables returned by RCSS in the LOCAL POOL.

Table 21. Variables Returned by RCSS in the LOCAL POOL

Variable Name	Type	Description
RCSSA(XXX)	Data	ASID of task
RCSSB(XXX)	Data	SQA used below the 16M line
RCSSC(XXX)	Data	SQA used above the 16M line
RCSSD(XXX)	Data	CSA used below the 16M line
RCSSE(XXX)	Data	CSA used above the 16M line
RCSSF(XXX)	Data	Total SQA used
RCSSG(XXX)	Data	Total CSA used
RCSSN(XXX)	Data	Name of task
RCSSROL1	Control	Number of output groups

RENQ: SYSPROG ENQUEUE Command

Use the RENQ EXEC to retrieve information about ENQUEUE conflicts.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RENQ)
```

There are no RENQ input parameters.

Following is a list of return codes from RENQ:

Return Code	Description
0	Request completed successfully
4	No enqueue conflicts
8	Security failure

Table 22 lists the variables returned by RENQ in the LOCAL POOL.

Table 22. Variables Returned by RENQ in the LOCAL POOL

Variable Name	Type	Description
RENQROL1	Control	Number of output lines
RENQSC&N	Data	Scope of ENQUEUE: step, system or systems
RENQGL&N	Data	Global or local
RENQM&N	Data	Major name of ENQUEUE
RENQR&N	Data	Resource name
RENQD&N	Data	Status: owns or wait
RENQT&N	Data	Type: SRC or EXC
RENQAS&N	Data	ASID
RENQTI&N	Data	Time
RENQU&N	Data	Jobname
RENQSY&N	Data	SYSID
RENQRC&N	Data	Resource count (if reserve assocociated with ENQUEUE)
RENQUN&N	Data	Unit address (for reserves) or null
RENQVL&N	Data	Volume serial (for reserves) or null

RIO: System Input/Output Information

Use the RIO EXEC to find all outstanding non-TP I/Os.

An optional parameter can be passed that limits the I/O information to the UCB which is passed by the caller.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RIO UCB) WAIT(YES)
```

Table 23 lists the RIO parameters.

Table 23. RIO Parameters

Parameter	Required?	Description
UCB	No	Device address

Following is a list of return codes from RIO:

Return Code	Description
0	Request successfully completed
4	No outstanding I/O for device
8	No outstanding I/O in system or security failure

Table 24 lists the variables returned by RIO in the LOCAL POOL.

Table 24. Variables Returned by RIO in the LOCAL POOL

Variable Name	Type	Description
RIOAL(XXX)	Data	Allocations
RIODV(XXX)	Data	Driver
RIOIA(XXX)	Data	IOQ address
RIOJN(XXX)	Data	Jobname
RIOOP(XXX)	Data	Opens
RIOPD(XXX)	Data	Paging device
RIOROL1	Control	Number of output lines
RIORV(XXX)	Data	Reserves
RIOUA(XXX)	Data	Unit address
RIOVS(XXX)	Data	Volume serial

RMDE: Device Monitoring

Use the RMDE EXEC to determine I/O bottlenecks in the system.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RMDE UCB TIME) WAIT(YES)
```

Table 25 lists the RMDE parameters.

Table 25. RMDE Parameters

Parameter	Required?	Description
UCB	No	Device address or a range of device addresses to monitor; the default is all devices
TIME	No	Length of time (in seconds) to monitor; the default is 15 seconds

Following is a list of return codes from RMDE:

Return Code	Description
4	All devices are less than 1 percent busy
8	Security failure
16	No specified devices are online

Table 26 lists the variables returned by RMDE in the LOCAL POOL.

Table 26. Variables Returned by RMDE in the LOCAL POOL

Variable Name	Type	Description
RMDEA(XXX)	Data	ACYL
RMDEB(XXX)	Data	Device busy
RMDEC(XXX)	Data	Connect time
RMDED(XXX)	Data	Disconnect time
RMDEP(XXX)	Data	Pend time
RMDEQ(XXX)	Data	Q length
RMDER(XXX)	Data	Rate
RMDEROL1	Control	Number of output lines
RMDES(XXX)	Data	Seek
RMDEU(XXX)	Data	Device UCB address
RMDEV(XXX)	Data	Volume serial

RMON: Address Space Monitoring

Use the RMON EXEC to monitor an address space to determine if its status is wait state, looping, or running normally.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RMON TASKNAME) WAIT(YES)
```

Table 27 lists the RMON parameters.

Table 27. RMON Parameters

Parameter	Required?	Description
TASKNAME	Yes	Name of task to monitor

Following is a list of return codes from RMON:

Return Code	Description
8	Security failure
16	Task name either not specified or not found

Table 28 list the variables returned by RMON in the LOCAL POOL.

Table 28. Variables Returned by RMON in the LOCAL POOL

Variable Name	Type	Description
RMONCPA	Data	CPU accumulated in last 30 seconds
RMONCPT	Data	Total CPU used
RMONDP	Data	Priority in page range notation
RMONEXA	Data	EXCPs accumulated in last 30 seconds
RMONEXT	Data	Total EXCPs
RMONHP	Data	Priority in decimal
RMONJN	Data	Task name
RMONNU	Data	Task number:sup
RMONPAA	Data	Pages accumulated in last 30 seconds
RMONPAT	Data	Total pages
RMONPG	Data	Performance group
RMONPP	Data	Performance period
RMONSN	Data	Step name
RMONSUA	Data	Service unit accumulated in last 30 seconds
RMONSUT	Data	Total service units
RMONTT	Data	Task type (TSU STC JOB)*
Note: *Variable will have a null value if started before JES or started as SUB=MSTR.		

RMPA: Channel Path Monitoring

Use the RMPA EXEC to monitor the percentage of activity or imbalances of a channel path.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RMPA PATH TIME) WAIT(YES)
```

Table 29 lists the RMPA parameters.

Table 29. RMPA Parameters

Parameter	Required?	Description
PATH	No	Path or range of paths
TIME	No	Length of time (in seconds) to monitor

Following is a list of return codes from RMPA:

Return Code	Description
4	Possible error condition, verify path(s)
8	Security failure
16	Path(s) specification error

Table 30 lists the variables returned by RMPA in the LOCAL POOL.

Table 30. Variables Returned by RMPA in the LOCAL POOL

Variable Name	Type	Description
RMPAB(XXX)	Data	Percent busy
RMPAP(XXX)	Data	Channel path
RMPAROL1	Control	Number of output lines

RMTP: Monitor Pending Mounts

Use the RMTP EXEC to find the volume serial number, UCBs, device types, and address spaces waiting for either tape or DASD mounts.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RMTP) WAIT(YES)
```

There are no RMTP input parameters.

Following is a list of return codes from RMTP:

Return Code	Description
4	More than 50 outstanding mounts
8	Security failure
16	No mounts pending

Table 31 lists the variables returned by RMTP in the LOCAL POOL.

Table 31. Variables Returned by RMTP in the LOCAL POOL

Variable Name	Type	Description
RMTPJ(XXX)	Data	Jobname that requested mount
RMTPT(XXX)	Data	Type of unit
RMTPU(XXX)	Data	Unit address of mount
RMTPV(XXX)	Data	Volser of requested volume

RPAG: System Wide Paging Information

Use the RPAG EXEC to find paging or demand paging rates.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RPAG) WAIT(YES)
```

There are no RPAG input parameters.

Following is a list of return codes from RPAG:

Return Code	Description
8	Security failure

Table 32 lists the variables returned by RPAG in the LOCAL POOL.

Table 32. Variables Returned by RPAG in the LOCAL POOL

Variable Name	Type	Description
RPAGCSA	Data	CSA paging rate
RPAGDMND	Data	Demand paging rate
RPAGLPA	Data	LPA paging rate
RPAGRATE	Data	Total paging rate
RPAGRCLM	Data	Page reclaim rate
RPAGSWAP	Data	Swap paging rate
RPAGTIME	Data	Elapsed time since counters last cleared
RPAGVIO	Data	VIO paging rate

RPRO: Monitor Progress of an Address Space

Use the RPRO EXEC to report on the progress of a task.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RPRO TASKNAME) WAIT(YES)
```

Table 33 lists the RPRO parameters.

Table 33. RPRO Parameters

Parameter	Required?	Description
TASKNAME	Yes	Name of task on which to report progress

Following is a list of return codes from RPRO:

Return Code	Description
0	Request completed successfully
4	Task not found
8	Security failure
16	Task not specified

Table 34 lists the variables returned by RPRO in the LOCAL POOL.

Table 34. Variables Returned by RPRO in the LOCAL POOL

Variable Name	Type	Description
RPROCL	Data	Job class
RPROCS	Data	Current step number
RPRODPTY	Data	Decimal version
RPROJD	Data	Job start date
RPROLCPU	Data	CPU limit
RPROMSGC	Data	Message class
RPROMSGL	Data	Message level
RPRONAME	Data	JOBNAME
RPRONUMB	Data	JES2 job number
RPROPCPU	Data	Percentage CPU used
RPROPG	Data	Performance group
RPROPGM	Data	Program name
RPROPGNM	Data	Programmer name
RPROPP	Data	Performance period
RPROPRTY	Data	Dispatching priority

Table 34. Variables Returned by RPRO in the LOCAL POOL (Continued)

Variable Name	Type	Description
RPRORR	Data	Region requested
RPRORU	Data	Region used
RPROSCPU	Data	Step total CPU
RPROSRB	Data	SRB time used (step)
RPROSS	Data	Step start time
RPROST	Data	Address space start time
RPROSTEP	Data	Currently executing step
RPROTCB	Data	TCB time used (step)
RPROTS	Data	Total steps
RPROTYPE	Data	Type of job (STC TSO JOB)
RPROVUA	Data	Virtual used above line
RPROVUB	Data	Virtual used below line

RREP: Retrieve WTOR IDs

The RREP EXEC can be used to retrieve the number and text of all outstanding WTORs.

The RREP EXEC can be used to retrieve the number and text of 10 outstanding WTORs.

If the EXEC receives more then ten WTORs, it ends with a retrurn code of 16, sets RREPROL1 to 0, issues an error message (REP101E) and deletes all data variables from the local pool. If you need to process or examine more then 10 WTORS, use the RREPRX EXEC (see page 453).

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RREP) WAIT(YES)
```

Table 35 lists the RREP parameters.

Table 35. RREP Parameters

Parameter	Required?	Description
SYSTEM	Yes	System ID for which to gather data. The value ALL means the same as no specification. All replies from a sysplex will be returned.

Following is a list of return codes from RREP:

Return Code	Description
8	Security failure
16	Received more than 10 WTORs

Table 36 lists the variables returned by RREP in the LOCAL POOL.

Table 36. Variables Returned by RREP in the LOCAL POOL

Variable Name	Type	Description
RREPI(XXX)	Data	Reply number
RREPN(XXX)	Data	JES number of task that issued WTOR
RREPROL1	Control	Number of output lines
RREPT(XXX)	Data	type of task that issued WTOR (STC TSU JOB)
RREP1(XXX) THROUGH RREP9(XXX)	Data	First nine words of message
RREP10(XXX) THROUGH RREP12(XXX)	Data	Extended to twelth word of the message

RREPRX: Retrieve WTOR IDs

The RREPRX EXEC can be used to retrieve the number and text of all outstanding WTORs.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RREPRX) WAIT(YES)
```

Table 37 lists the RREPRX parameters.

Table 37. RREPRX Parameters

Parameter	Required?	Description
SYSTEM	No	System ID for which to gather data. The value ALL means the same as no specification. All replies from a sysplex will be returned. If no system ID is given, the default if the local system ID.

Following is a list of return codes from RREPRX:

Return Code	Description
0	Normal completion
4	No outstanding replies found
8	Security failure
12	Command timed out
16	Error detected. RREPRX EXEC exited.

Table 38 lists the variables returned by RREPRX in the LOCAL POOL.

Table 38. Variables Returned by RREPRX in the LOCAL POOL

Variable Name	Type	Description
RREPROL1	Control	Number of output lines
RREPS.x	Data	The system ID of the issuer, x = line number
RREPT.x	Data	Type of task, STC, TSU, JOB, ASID or NULL for JES3. NULL if task was started before JES
RREPN.x	Data	JES jobnumber of the task or jobname if JES3 Null if task was started before JES
RREPI.x	Data	The reply number
RREP.x.1-12	Data	The first 12 words of the message

RRES: Retrieve Outstanding Reserves

Use the RRES EXEC to retrieve information on outstanding device reserves.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RRES) WAIT(YES)
```

There are no RRES input parameters.

Following is a list of return codes from RRES:

Return Code	Description
0	Request completed successfully
4	No outstanding reserves
8	Security failure

Table 39 lists the variables returned by RRES in the LOCAL POOL.

Table 39. Variables Returned by RRES in the LOCAL POOL

Variable Name	Type	Description
RRESROL1	Control	Number of output lines
RRESSC(XXX)	Data	Scope of reserve: systems
RRESGL(XXX)	Data	Global or local
RRESM(XXX)	Data	Major name of reserve
RRESN(XXX)	Data	Minor name of reserve
RRESSY(XXX)	Data	SYSID
RRESJ(XXX)	Data	Jobname
RRESAS(XXX)	Data	ASID
RRESC(XXX)	Data	Status: owns or wait
RREST(XXX)	Data	Type: SHR OR EXC
RRESRC(XXX)	Data	Reserve count
RRESS(XXX)	Data	Pend: yes or no
RRESU(XXX)	Data	Unit address
RRESV(XXX)	Data	Volume serial
RRESUR(XXX)	Data	Unit ready: no or null
RRESTI(XXX)	Data	Time

RRSM: Real Storage Management Information

The RRSM EXEC returns basic real storage information on a system-wide basis. It also can produce a detail line for each address space in the system.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RRSM OPTS) WAIT(YES)
```

Table 40 lists the RRSM parameters.

Table 40. RRSM Parameters

Parameter	Required?	Description
OPTS	No	If specified, must be MAP
SORTFLD	No	NAME, ASID, FRAMES, FIXED, <16MB, LSQA, WSS or PERCENT

Note: All values for SORTFLD (with the exception of NAME and ASID) will return only record with the 10 highest values for that field.

Following is a list of return codes from RRSM:

Return Code	Description
8	Security failure
16	Other than MAP parameter specified

Table 41 lists the variables returned by RRSM in the LOCAL POOL.

Table 41. Variables Returned by RRSM in the LOCAL POOL

Variable Name	Type	Description
RRSMA(XXX)	Data	ASID of task
RRSMAFRM	Data	Available frames (free)
RRSMB(XXX)	Data	Fixed below 16 MB line
RRSMB16M	Data	Fixed frames below 16 MB line
RRSMCFFR	Data	Common fixed frames
RRSMCFRM	Data	Common frames
RRSMFFRM	Data	Fixed frames
RRSML(XXX)	Data	LSQA frames
RRSMN(XXX)	Data	Name of task
RRSMNFRM	Data	Nucleus frames
RRSMOFRM	Data	Online frames
RRSMP(XXX)	Data	Percentage of online frames allocated

Table 41. Variables Returned by RRSM in the LOCAL POOL (Continued)

Variable Name	Type	Description
RRSMPFFR	Data	Private fixed frames
RRSMPFRM	Data	Private frames
RRSMS(XXX)	Data	Storage frames
RRSMSFFR	Data	SQA fixed frames
RRSMW(XXX)	Data	Working set size

RSPA: Retrieve DASD Space Information

The RSPA service can be used to retrieve free space and contiguous free space information. The following statement shows the format of the call from an EXEC:

```
IMFEXEC SELECT EXEC(RSPA DEVICE MOUNTED ONLINE SIZE) WAIT(YES)
```

The following table lists the RSPA parameters.

Table 42. RSPA Parameters

Parameter	Required?	Description
DEVICE	No	If specified, must be mounted PUBLIC. PRIVATE will select devices mounted PRIVATE. **Note:** DEVICE is required if MOUNTED or ONLINE parameters are specified. If ONLINE = ALL is specified, the DASD information returned includes all OFFLINE devices as well as those ONLINE devices that meet the selection criteria.
ONLINE	No	Default is ONLINE. If specified, must be: **ONLINE** Selects ONLINE devices only. **OFFLINE** Selects OFFLINE devices only. ONLINE = OFFLINE is mutually exclusive with the MOUNTED and SIZE parameters. MOUNTED and SIZE must be NULL if ONLINE = OFFLINE is specified. **ALL** Selects OFFLINE and ONLINE devices. If ONLINE = ALL is specified, the DASD information returned includes all OFFLINE devices as well as those ONLINE devices that meet the selection criteria.

Table 42. RSPA Parameters (Continued)

Parameter	Required?	Description
SIZE	No	Default is 0. ONLINE = OFFLINE is mutually exclusive with the MOUNTED and SIZE parameters. MOUNTED and SIZE must be NULL if ONLINE = OFFLINE is specified. If specified, must be a numeric value between 0 and 999. The RSPA service will return information for ONLINE DASD volumes that have more than nnn free space cylinders.
SORTFLD	No	Unit, VOLSER, FREE or CONTIG

Following is a list of return codes from RSTA:

Return Code	Description
0	The returned information includes at least one complete DASD Unit entry.
4	The returned information contains at least one incomplete (OFFLINE) DASD unit entry (VOLUME = UNKNWN and all DASD space fields = 0).
8	No information was returned or security failure An informational message RSPnnnx is displayed that specifies why no information is returned. RSPAROL1 is set to zero.
12	An unexpected error message was returned by the SYSPROG SPACE service. Message RSP015I displays the error message issued by the SYSPROG SPACE service. RSPAROL1 is set to zero.
16	Invalid parameters passed to the RSPA service EXEC. RSPAROL1 is set to zero.

Table 43 lists the variables returned by RSPA in the LOCAL POOL.

Table 43. Variables Returned by RSPA in the LOCAL POOL

Variable Name	Type	Description
RSPAROL1	Data	Number of entries in the returned data array.
RSPAU(XXX)	Data	Unit number (device address)
RSPAV(XXX)	Data	Volume Name Value is set to UNKNWN for OFFLINE devices.
RSPAS(XXX)	Data	Device Mount Attribute: OFFLINE OR PRV (PRIVATE) STR (STORAGE) PUB (PUBLIC)
RSPAC(XXX)	Data	Number of free cylinders (0 if OFFLINE).
RSPAT(XXX)	Data	Number of free tracks (0 if OFFLINE).
RSPAG(XXX)	Data	Number of contiguous free cylinders (0 if OFFLINE).
RSPAH(XXX)	Data	Number of contiguous free tracks (0 if OFFLINE).

RSTA: Retrieve Status of an Address Space

The RSTA EXEC retrieves the status of any or all tasks in the system.

If a task name is specified but not enabled in the system, RSTA sets a return code of 4, making it easy to determine if a task is enabled.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RSTA TASKNAME) WAIT(YES)
```

Table 44 lists the RSTA parameters.

Table 44. RSTA Parameters

Parameter	Required?	Description
TASKNAME	No	Specific task on which to retrieve status information

Following is a list of return codes from RSTA:

Return Code	Description
0	Task is enabled
4	Task is not enabled
8	Security failure
12	Service timed out in interface
16	Invalid parameter specified

Table 45 lists the variables returned by RSTA in the LOCAL POOL.

Table 45. Variables Returned by RSTA in the LOCAL POOL

Variable Name	Type	Description
RSTAA(XXX)	Data	Address space ID
RSTAC(XXX)	Data	CPU time used
RSTAF(XXX)	Data	Real frame count
RSTAG(XXX)	Data	Performance group
RSTAN(XXX)	Data	Name of task
RSTAP(XXX)	Data	Performance period
RSTAQ(XXX)	Data	Dispatching queue
RSTAW(XXX)	Data	Working set size
RSTA1(XXX)	Data	Status 1
RSTA2(XXX)	Data	Status 2
RSTA2(ROL1)	Control	Number of output lines

RSYS: System Dump Data Sets Information

The RSYS EXEC retrieves information on all the system dump data sets.

An example of the use of the RSYS EXEC is given in the MSU005C EXEC.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RSYS) WAIT(YES)
```

There are no RSYS input parameters.

Following is a list of return codes from RSYS:

Return Code	Description
4	Task is not enabled
8	Security failure

Table 46 lists the variables returned by RSYS in the LOCAL POOL.

Table 46. Variables Returned by RSYS in the LOCAL POOL

Variable Name	Type	Description
RSYSD(XXX)	Data	Day of month data set was filled
RSYSN(XXX)	Data	Name of full dump data set
RSYSROL1	Control	Number of output lines
RSYST(XXX)	Data	Time data set was filled
RSYSG(XXX)	Data	Date of dump in MMM DD YYYY format
RSYSI(XXX)	Data	Title of the dump
RSYSS(XXX)	Data	Source of the dump

RTPI: Teleprocessing Input/Output Information

The RTPI EXEC retrieves information on teleprocessing I/O (TP I/O) in the system. If a parameter is passed, the information returned is only for the specified resource; otherwise, the information for all TP I/O in the system is returned.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(RTPI OPTS) WAIT(YES)
```

Table 47 lists the RTPI parameters.

Table 47. RTPI Parameters

Parameter	Required?	Description
OPTS	No	UCB or volume serial number

Following is a list of return codes from RTPI:

Return Code	Description
0	Request completed successfully
4	No outstanding I/O
8	Security failure

Table 48 lists the variables returned by RTPI The following table lists the variables returned by RTPI. in the LOCAL POOL.

Table 48. Variables Returned by RTPI in the LOCAL POOL

Variable Name	Type	Description
RTPIA(XXX)	Data	Allocations
RTPID(XXX)	Data	Driver
RTPII(XXX)	Data	IOQ address
RTPIJ(XXX)	Data	Jobname using device
RTPIO(XXX)	Data	Opens
RTPIR(XXX)	Data	Reserves
RTPIROL1	Control	Number of output lines
RTPIU(XXX)	Data	UCB of device
RTPIV(XXX)	Data	VOLSER of device (may be blank)

RTSU: Information on TSO Users

The RTSU EXEC retrieves information on TSO users in the system. If a parameter is passed, the information returned is only for the specified TSO user; otherwise, information on all TSO users is returned.

The following statement shows the format of the call from EXEC:

```
IMFEXEC SELECT EXEC(RTSU OPTS) WAIT(YES)
```

Table 49 lists the RTSU parameters.

Table 49. RTSU Parameters

Parameter	Required?	Description
OPTS	No	TSO User ID

Following is a list of return codes from RTSU:

Return Code	Description
0	Request completed successfully
4	Specified USER ID not found
8	Security failure
16	Specified USER ID greater than seven characters

Table 50 lists the variables returned by RTSU in the LOCAL POOL.

Table 50. Variables Returned by RTSU in the LOCAL POOL

Variable Name	Type	Description
RTSUA(XXX)	Data	ASID of TSO user
RTSUL(XXX)	Data	TCAM line number (0S for VTAM)
RTSUN(XXX)	Data	Node name used EXEC: N=LOOP CTR T=TSO LINE COUNTER
RTSUROL1	Control	Number of output lines
RTSUS(XXX)	Data	System (TCAM OR VTAM)
RTSUU(XXX)	Data	TSO user ID
RTSUUSER	Control	Number OF TSO users logged on

@TIMER: Interface to Timer Queues

The @TIMER EXEC provides a common interface to the timer queue functions.

Purpose

Use the @TIMER EXEC to introduce time-initiated EXECs (also called timer elements) into AutoOPERATOR. Timer elements are EXECs which are automatically invoked at user-specified intervals for a user-specified period of time. As an option, timer elements can also be invoked one at a time. Refer to the *MAINVIEW AutoOPERATOR Basic Automation Guide* for more information about time-initiated EXECs.

Function

Use @TIMER to add or delete timer elements from the system. The specific function is controlled by the input parameters you use. Table 51 lists the @TIMER parameters.

Deleting a timer element has 2 separate processes. The first issues an `IMFEXEC IMFC PRG=CALLX timername` statements where `timername` is the name of the timer element to be deleted from the system. The delete is performed by the AutoOPERATOR interval services task.

The second deletes the PROFILE pool variables that contain the parameters necessary to invoke and maintain the timer element.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC[@TIMER FUNC(ADD|DEL) PROC(execname ) +
   TOD(HH:MM:SS) [or NEXTTIME(MMMM)] INTERVAL(HHcolon.MM:SS) +
   GOODFOR(MMMMM) [or RETAIN(YES|NO)] TARGETSS(subsys ID)
        REPLACE(YES|NO) +
   STOPTIME(HH:MM:SS) DEBUG(YES|NO|TRACE)] WAIT(YES)
```

All parameters use keywords; the value must be enclosed in parentheses following the keyword. Table 51 lists the @TIMER parameters.

Table 51. @TIMER Parameters

Parameter	Required?	Description
FUNC	Yes	Function to perform Valid values are ADD or DEL(ete).
PROC	Yes	Name of an EXEC to schedule
TOD	Yes	Time at which to schedule the process A valid time must be entered in Hours:Minutes:Seconds format (HH:MM:SS). **Note:** The NEXTTIME and TOD parameters are mutually exclusive and cannot be used together.

Table 51. @TIMER Parameters (Continued)

Parameter	Required?	Description
NEXTTIME	Yes	The amount of time (from current time) to schedule the process A valid time must be entered in MMMM format where MMMM is an up-to 4 digit number. The NEXTTIME parameter MMMM should not result in a start time that is greater than 24 hours. Therefore NEXTTIME should not be greater than 1440. **Note:** The NEXTTIME and TOD parameters are mutually exclusive and cannot be used together.
INTERVAL	No	Repetition interval Specify a time in HH:MM:SS format that will cause the the function to be repeated at set intervals.
GOODFOR	No	The amount of time this timer element is good for A valid time must be entered in MMMM format where MMMM is an up-to 4 digit number. This value is used by the AutoOPERATOR Sample Catch-Up Solution to determine whether or not this element should be re-instated after an AutoOPERATOR shutdown. The GOODFOR time is converted into a time format used by Catch-Up processing and compared against the originally scheduled and current times to determine reinstatability. **Note:** The GOODFOR and RETAIN parameters are mutually exclusive and cannot be used together.
RETAIN	No	Retain for catchup processing Valid values are YES or NO When RETAIN(YES) is specified, it is the number of minutes beyond the time specified in TOD in which catch up processing will be valid. **Note:** The GOODFOR and RETAIN parameters are mutually exclusive and cannot be used together.
TARGETSS	No	The subsystem ID (SYSID) of the target for execution

Table 51. @TIMER Parameters (Continued)

Parameter	Required?	Description
REPLACE	No	Used on an ADD request to determine whether or not a request should replace a current timer element of the same name. Valid values are YES or NO.
STOPTIME	No	Time to stop scheduling the process: A valid time must be entered in Hours:Minutes:Seconds format (HH:MM:SS).
DEBUG	No	Causes the display of debugging messages to be issued to the BBI-SS PAS Journal during the execution of @TIMER. Valid values are YES, NO or TRACE. YES causes informational messages to be issued by @TIMER in the format SOLnnnt, NO (default) causes no debugging messages to be written to the BBI-SS PAS Journal. TRACE causes the SOLnnnt messages, along with CONTROL CON SYM and IMFEXEC CNTL LIST trace output to be written to the BBI-SS PAS Journal.

Processing

When an ADD request is received, @TIMER determines if the timer element already exists. If the element does not exist, a LIST variable is created to define and maintain the element. An IMFEXEC IMFC SET REQ=CALLX command is also built and sent to the interval services task. The interval services task then prepares the internal control block that defines this request and the timer element is ready to execute at the specified time.

If the timer element does exist (meaning that the LIST variable is present in the PROFILE variable pool), @TIMER determines if this element will be replaced by checking the input parameter REPLACE(). If the element is to be replaced, @TIMER determines if the element is pending delete (another request has already started the delete process) and if not, issues the command to delete the element to the interval services task.

If a delete is pending, an SOL230E message is issued and @TIMER waits up to 5 minutes for the delete to finish. After 5 minutes, if the pending delete has not taken place, an SOL242E message is issued and the add request is discarded.

If the element is not to be replaced, @TIMER determines whether or not the element is pending delete by another request. If the element is pending delete, @TIMER waits 30 seconds for the delete to complete. If that delete does not occur, an SOL240W message is issued and @TIMER deletes the timer element list variable and proceeds with the add request.

If the element is not pending delete, an SOL222E message is issued and the request is discarded.

When a timer element delete is initiated, @TIMER first determines if the timer element exists. If it does exist, @TIMER issues the command to delete the element to the interval services task, sets the element to pending delete status and exits. The second process of the delete then takes place as another invocation of @TIMER deletes the PROFILE variable that defines the timer element.

If the timer element does not exist, an SOL229E message is issued and @TIMER exits. If the timer element is pending delete from another request, an SOL230E messages is issued and @TIMER exits.

Following is a list of return codes from @TIMER:

Return Code	Description
0	Request completed successfully
4	Duplicate element found, ADD failed
8	Element not found, DELETE failed
16	Parameter error; refer to error message for further clarification

JES2DI: Retrieve Initiator Information

The JES2DI EXEC can be used to retrieve information on JES2 initiators.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(JES2DI 1 10) WAIT(YES)
```

Table 52 lists the JES2DI parameters.

Table 52. JES2DI Parameters

Parameter	Required?	Description
BINIT	No	Beginning initiator number for the display
EINIT	No	Ending initiator number for the display

Following is a list of return codes from JES2DI:

Return Code	Description
0	Request completed successfully
8	Parameter errors
12	Command not completed due to timeout

Table 53 lists the variables returned by JES2DI in the LOCAL POOL.

Table 53. Variables Returned by JES2DI in the LOCAL POOL

Variable Name	Type	Description
JES2NOL	Control	Number of output lines
J2DINM(XX)	Data	Initiator number
J2DIST(XX)	Data	Status of the init
J2DIJNM(XX)	Data	Currently executing job number
J2DICLS(XX)	Data	Assigned classes

JES2DQ: Retrieve Execution Queue Information

Use the JES2DQ EXEC to retrieve information on JES2's execution queues.

The following statement shows the format of the call from EXEC:

```
IMFEXEC SELECT EXEC(JES2DQ) WAIT(YES)
```

There are no input parameters to JES2DQ.

Following is a list of the return codes from JES2DQL:

Return Code	Description
0	Request completed successfully
12	Command not completed due to timeout

Table 54 lists the variables returned by JES2DQ.

Table 54. Variables Returned by JES2DQ in the LOCAL POOL

Variable	Type	Description
JES2NOL	Control	Number of output lines
J2DQJNB(XX)	Data	Number of jobs in queue
J2DQCLS(XX)	Data	Class for this queue
J2DQSYS(XX)	Data	SYSID for this queue

CNVSECS: Convert HH:MM:SS Format to Seconds

Use the CNVSECS EXEC convert time in the HH:MM:SS format to seconds.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(CNVSECS TIMEIN) WAIT(YES)
```

Table 55 lists the CNVSECS parameters.

Table 55. CNVSECS Parameters

Parameter	Required?	Description
TIMEIN	Yes	Time in HH:MM:SS format

Following is a list of return codes from CNVSECS:

Return Code	Description
0	Request completed successfully
8	Input parameter not in HH:MM:SS format
12	Input parameter not specified

Table 56 lists the variables returned by CNVSECS in the LOCAL POOL.

Table 56. Variables Returned by CNVSECS in the LOCAL POOL

Variable	Type	Description
SECSOUT	Data	Output time in seconds

CNVTIME: Convert Time in Seconds to HH:MM:SS

Use the CNVTIME EXEC to convert time in seconds to HH:MM:SS format.

The following statement shows the format of the call from the EXEC:

```
IMFEXEC SELECT EXEC(CNVTIME SECSIN) WAIT(YES)
```

Table 57 lists the CNVTIME parameters.

Table 57. CNVTIME Parameters

Parameter	Required?	Description
SECSIN	Yes	Time in seconds

Following is a list of return codes from CNVTIME:

Return Code	Description
0	Request completed successfully
12	Input parameter not specified

Table 58 lists the variables returned by CNVTIME in the LOCAL POOL.

Table 58. Variables Returned by CNVTIME in the LOCAL POOL

Variable	Type	Description
TIMEOUT	Data	Output time in HH:MM:SS format
CNVTDAY	Data	Number of days the output goes past the day boundary

Appendix A. SYSPROG EXEC Cross-Reference

Table 59 cross-references the SYSPROG service EXECs and variables. The table is sorted alphabetically by the SYSPROG service field name in the first column.

To use the table, review the left column to find the type of SYSPROG service EXEC information you need. For each SYSPROG service field, there is a description of the EXEC variable and the EXEC variable name.

The first four positions of the variable name indicate the EXEC name. For example, the first variable listed, RMDEA(XXX) is used in the utility EXEC RMDE. Refer to RMDE EXEC for a full set of services available and the RMDEA(XXX) variable.

Table 59. SYSPROG Service EXEC and Variable Cross-Reference

SYSPROG Service Name	EXEC Variable Description	Variable Name
ACYL	Determine I/O System Bottlenecks	RMDEA(XXX)
ADDRESS OF VECTOR TABLE	Storage/Paging/Slots	RASMVTA
ADDRESS SPACE ID	Return Status of Task(s)	RSTAA(XXX)
ADDRESS SPACE START TIME	Progress Report on Task	RPROST
ALLOCATIONS	Virtual Storage Address Space	RIOAL(XXX)
ALLOCATIONS	Interface to SYSPROG service TPIO	RTPIA(XXX)
ASID OF TASK	Virtual Storage Address Space	RCSSA(XXX)
ASID OF TASK	Basic Real Storage Information	RRSMA(XXX)
ASID OF TSO USER	TSO User System Information	RTSUA(XXX)
AVAILABLE FRAMES(FREE)	Basic Real Storage Information	RRSMAFRM
AVAILABLE SLOTS	Storage/Paging/Slots	RASMSLAV
BURST	Storage/Paging/Slots	RASMB(XXX)
CHANNEL PATH	Monitor Channel Path	RMPAP(XXX)
COMMON FIXED FRAMES	Basic Real Storage Information	RRSMCFFR
COMMON FRAMES	Basic Real Storage Information	RRSMCFRM
CONNECT TIME	Determine I/O System Bottlenecks	RMDEC(XXX)
CPU ACCUMULATED IN LAST 30 SECONDS	Monitor Address Space	RMONCPA
CPU LIMIT	Progress Report on Task	RPROLCPU
CPU NUMBER	CPU/TSO/MVS Overhead	RCPUC(XX)
CPU % BUSY	CPU/TSO/MVS Overhead	RCPUB(XX)
CPU TIME USED	Return Status of Task(s)	RSTAC(XXX)

Table 59. SYSPROG Service EXEC and Variable Cross-Reference (Continued)

SYSPROG Service Name	EXEC Variable Description	Variable Name
CSA PAGING RATE	Paging/Demand Paging Information	RPAGCSA
CSA USED ABOVE THE 16M LINE	Virtual Storage Address Space	RCSSE(XXX)
CSA USED BELOW THE 16M LINE	Virtual Storage Address Space	RCSSD(XXX)
CURRENT CONDITION	SYSPROG service RESERVE Interface	RRESC(XXX)
CURRENT STEP NUMBER	Progress Report on Task	RPROCS
CURRENTLY EXECUTING STEP	Progress Report on Task	RPROSTEP
DATE DATA SET WAS FILLED	Interface to SYSDUMP Service	RSYSD(XXX)
DECIMAL VERSION	Progress Report on Task	RPRODPTY
DEMAND PAGING RATE	Paging/Demand Paging Information	RPAGDMND
DEVICE BUSY	Determine I/O System Bottlenecks	RMDEB(XXX)
DEVICE UCB ADDRESS	Determine I/O System Bottlenecks	RMDEU(XXX)
DISCONNECT TIME	Determine I/O System Bottlenecks	RMDED(XXX)
DISPATCHING PRIORITY	CPU/TSO/MVS Overhead	RCPUD(XX)
DISPATCHING PRIORITY	Progress Report on Task	RPROPRTY
DISPATCHING QUEUE	Return Status of Task(s)	RSTAQ(XXX)
DRIVER	Find ALL Outstanding Non-TP I/O	RIODV(XXX)
DRIVER	Interface to SYSPROG service TPIO	RTPID(XXX)
ELAPSED TIME SINCE CTRS LAST CLEARED	Paging/Demand Paging Information	RPAGTIME
EXCPS ACCUMULATED IN LAST 30 SECONDS	Monitor Address Space	RMONEXA
FIRST 12 WORDS OF MESSAGE	SYSPROG service REPLIES Interface	RREP1(XXX) THROUGH RREP9(XXX)
FIRST 12 WORDS OF MESSAGE	SYSPROG service REPLIES Interface	RREP1.X.1-12
FIXED BELOW 16MB LINE	Basic Real Storage Information	RRSMB(XXX)
FIXED FRAMES	Basic Real Storage Information	RRSMFFRM
FIXED FRAMES BELOW 16MB LINE	Basic Real Storage Information	RRSMB16M
FIXED STORAGE FRAMES	Basic Real Storage Information	RRSMF(XXX)
FREE SLOTS	Storage/Paging/Slots	RASMB(XXX)
IOQ ADDRESS	Find ALL Outstanding Non-TP I/O	RIOIA(XXX)
IOQ ADDRESS	Interface to SYSPROG service TPIO	RTPII(XXX)

Table 59. SYSPROG Service EXEC and Variable Cross-Reference (Continued)

SYSPROG Service Name	EXEC Variable Description	Variable Name
IPL TYPE	Storage/Paging/Slots	RASMIPLT
JES NUMBER OF TASK THAT ISSUED WTOR	SYSPROG service REPLIES Interface	RREPN(XXX)
JES NUMBER OF TASK THAT ISSUED WTOR	SYSPROG service REPLIES Interface	RREPN.X
JES2 JOB NUMBER	Progress Report on Task	RPRONUMB
JOB CLASS	Progress Report on Task	RPROCL
JOBNAME	Storage/Paging/Slots	RASMJ(XXX)
JOBNAME	Find ALL Outstanding Non-TP I/O	RIOJN(XXX)
JOBNAME	Progress Report on Task	RPRONAME
JOBNAME THAT REQUESTED MOUNT	Tape or DASD Mount Requests	RMPTJ(XXX)
JOBNAME USING DEVICE	Interface to SYSPROG service TPIO	RTPIJ(XXX)
JOBNAME WITH POSSIBLE RESERVE	SYSPROG service service RESERVE Interface	RRESJ(XXX)
LABEL OF VOLUME	Storage/Paging/Slots	RASML(XXX)
LARGEST SLOT USER	Storage/Paging/Slots	RASMSUSRX
LPA PAGING RATE	Paging/Demand Paging Information	RPAGLPA
LSQA FRAMES	Basic Real Storage Information	RRSML(XXX)
MAJOR NAME OF RESERVE	SYSPROG service service RESERVE Interface	RRESM(XXX)
MESSAGE CLASS	Progress Report on Task	RPROMSGC
MESSAGE LEVEL	Progress Report on Task	RPROMSGL
MINOR NAME OF RESERVE	SYSPROG service RESERVE Interface	RRESN(XXX)
NAME OF FULL DUMP DATA SET	Interface to SYSDUMP Service	RSYSN(XXX)
NAME OF JOB	CPU/TSO/MVS Overhead	RCPUN(XX)
NAME OF TASK	Virtual Storage Address Space	RCSSN(XXX)
NAME OF TASK	Basic Real Storage Information	RRSMN(XXX)
NAME OF TASK	Return Status of Task(s)	RSTAN(XXX)
NODE NAME EXEC N=LOOP CTR T=TSO LINE CTR	TSO User Information	RTSUN(XXX)
NON-VIO SLOTS	Storage/Paging/Slots	RASMN(XXX)
NUCLEUS FRAMES	Basic Real Storage Information	RRSMNFRM
NUMBER OF CPU RELATED OUTPUT GROUPS	CPU/TSO/MVS Overhead	RCPUROL3

Table 59. SYSPROG Service EXEC and Variable Cross-Reference (Continued)

SYSPROG Service Name	EXEC Variable Description	Variable Name
NUMBER/JOB RELATED OUTPUT VARIABLE GRPS	CPU/TSO/MVS Overhead	RCPUROL2
NUMBER OF OUTPUT LINES	Monitor Channel Path	RMPAROL1
NUMBER OF OUTPUT GROUPS	Virtual Storage Address Space	RCSSROL1
NUMBER OF OUTPUT LINES	Find ALL Outstanding Non-TP I/O	RIOROL1
NUMBER OF OUTPUT LINES	Determine I/O System Bottlenecks	RMDEROL1
NUMBER OF OUTPUT LINES	SYSPROG service REPLIES Interface	RREPROL1
NUMBER OF OUTPUT LINES	Interface to SYSDUMP Service	RSYSROL1
NUMBER OF OUTPUT LINES	Interface to SYSPROG service TPIO	RTPIROL1
NUMBER OF OUTPUT LINES	TSO User System Information	RTSUROL1
NUMBER OF SYSPROG SERVICE OUTPUT LINES	Storage/Paging/Slots	RASMROL1
NUMBER OF TSO USERS LOGGED ON	TSO User System Information	RTSUUSER
ONLINE FRAMES	Basic Real Storage Information	RRSMOFRM
OPENS	Find ALL Outstanding Non-TP I/O	RIOOP(XXX)
OPENS	Interface to SYSPROG service TPIO	RTPIO(XXX)
PAGE RECLAIM RATE	Paging/Demand Paging Information	RPAGRCLM
PAGES ACCUMULATED IN LAST 30 SECONDS	Monitor Address Space	RMONPAA
PAGING DEVICE	Find ALL Outstanding Non-TP I/O	RIOPD(XXX)
PEND TIME	Determine I/O System Bottlenecks	RMDEP(XXX)
PERCENT BUSY	Monitor Channel Path	RMPAB(XXX)
PERCENTAGE AVAILABLE	Storage/Paging/Slots	RASMSLPC
PERCENTAGE CPU USED	Progress Report on Task	RPROPCPU
PERCENTAGE FREE	Storage/Paging/Slots	RASMP(XXX)
PERCENTAGE LARGEST USER IS HOLDING	Storage/Paging/Slots	RASMV(XXX)
PERCENTAGE OF CPU USED BY BATCH	CPU/TSO/MVS Overhead	RCPUBATP
PERCENTAGE OF CPU IN MVSOVERHEAD	CPU/TSO/MVS Overhead	RCPUMVSP
PERCENTAGE OF CPU USED BY TSO	CPU/TSO/MVS Overhead	RCPUTSOP
PERCENTAGE OF CPU	CPU/TSO/MVS Overhead	RCPUU(XX)

Table 59. SYSPROG Service EXEC and Variable Cross-Reference (Continued)

SYSPROG Service Name	EXEC Variable Description	Variable Name
PERCENTAGE OF ONLINE FRAMES ALLOCATED	Basic Real Storage Information	RRSMP(XXX)
PERFORMANCE GROUP	Monitor Address Space	RMONPG
PERFORMANCE GROUP	Progress Report on Task	RPROPG
PERFORMANCE GROUP	Return Status of Task(s)	RSTAG(XXX)
PERFORMANCE PERIOD	Monitor Address Space	RMONPP
PERFORMANCE PERIOD	Progress Report on Task	RPROPP
PERFORMANCE PERIOD	Return Status of Task(s)	RSTAP(XXX)
PRIORITY	CPU/TSO/MVS Overhead	RCPUP(XX)
PRIORITY IN APG RANGE NOTATION	Monitor Address Space	RMONDP
PRIORITY IN DECIMAL	Monitor Address Space	RMONHP
PRIVATE FIXED FRAMES	Basic Real Storage Information	RRSMPFFR
PRIVATE FRAMES	Basic Real Storage Information	RRSMPFRM
PROGRAM NAME	Progress Report on Task	RPROPGM
PROGRAMMER NAME	Progress Report on Task	RPROPGNM
Q LENGTH	Determine I/O System Bottlenecks	RMDEQ(XXX)
RATE	Determine I/O System Bottlenecks	RMDER(XXX)
REAL FRAME COUNT	Return Status of Task(s)	RSTAF(XXX)
REGION REQUESTED	Progress Report on Task	RPRORR
REGION USED	Progress Report on Task	RPRORU
REPLY NUMBER	SYSPROG service REPLIES Interface	RREPI(XXX)
REPLY NUMBER	SYSPROG service REPLIES Interface	RREPI.X
RESERVES	Find ALL Outstanding Non-TP I/O	RIORV(XXX)
RESERVES	Interface to SYSPROG service TPIO	RTPIR(XXX)
SECONDS OF CPU USED BY BATCH	CPU/TSO/MVS Overhead	RCPUBATT
SECONDS OF CPU IN MVS OVERHEAD	CPU/TSO/MVS Overhead	RCPUMSVO
SECONDS OF CPU USED	CPU/TSO/MVS Overhead	RCPUS(XX)
SECONDS OF CPU USED BY TSO	CPU/TSO/MVS Overhead	RCPUTSOT
SEEK	Determine I/O System Bottlenecks	RMDES(XXX)
SERVICE UNIT ACCUMM IN LAST 30 SEC.	Monitor Address Space	RMONSUA
SIZE OF DATA SET IN SLOTS	Storage/Paging/Slots	RASMS(XXX)

Table 59. SYSPROG Service EXEC and Variable Cross-Reference (Continued)

SYSPROG Service Name	EXEC Variable Description	Variable Name
SQA FIXED FRAMES	Basic Real Storage Information	RRSMSFFR
SQA USED ABOVE THE 16M LINE	Virtual Storage Address Space	RCSSC(XXX)
SQA USED BELOW THE 16M LINE	Virtual Storage Address Space	RCSSB(XXX)
SRB TIME USED(STEP)	Progress Report on Task	RPROSRB
START LINE NUMBER OF PAGE D/S INFORMATION	Storage/Paging/Slots	RASMROL2
START LINE NUMBER OF MAP INFORMATION	Storage/Paging/Slots	RASMROL3
STATUS OF RESERVE	SYSPROG service RESERVE Interface	RRESS(XXX)
STATUS 1(NSW LS PVL OUT GOI GOB ENQ IN GOO)	Status of Task(s)	RSTA1(XXX)
STATUS 2(LS MS LW CP)	Status of Task(s)	RSTA2(XXX)
STEP NAME	Monitor Address Space	RMONSN
STEP START TIME	Progress Report on Task	RPROSS
STEP TOTAL CPU	Progress Report on Task	RPROSCPU
STORAGE FRAMES	Basic Real Storage Information	RRSMS(XXX)
SWAP PAGING RATE	Paging/Demand Paging Information	RPAGSWAP
SYSTEM(TCAM OR VTAM)	TSO User System Information	RTSUS(XXX)
SYSTEM ID	The system ID of the issuer, x = line number	RREPS.X
TASK NAME	Monitor Address Space	RMONJN
TASK NUMBER	Monitor Address Space	RMONNU
TASK TYPE (TSU STC JOB)	Monitor Address Space	RMONTT
TCAM LINE NUMBER(0S FOR VTAM)	TSO User System Information	RTSUL(XXX)
TCB TIME USED(STEP)	Progress Report on Task	RPROTCB
TIME DATA SET WAS FILLED	Interface to SYSDUMP Service	RSYST(XXX)
TOTAL CPU USED	Monitor Address Space	RMONCPT
TOTAL CSA USED	Virtual Storage Address Space	RCSSG(XXX)
TOTAL EXCPS	Monitor Address Space	RMONEXT
TOTAL NUMBER OF OUTPUT LINES	CPU/TSO/MVS Overhead	RCPUROL1
TOTAL PAGES	Monitor Address Space	RMONPAT
TOTAL PAGING RATE	Paging/Demand Paging Information	RPAGRATE
TOTAL SERVICE UNITS	Monitor Address Space	RMONSUT

Table 59. SYSPROG Service EXEC and Variable Cross-Reference (Continued)

SYSPROG Service Name	EXEC Variable Description	Variable Name
TOTAL SLOTS	Storage/Paging/Slots	RASMTLSL
TOTAL SQA USED	Virtual Storage Address Space	RCSSF(XXX)
TOTAL STEPS	Progress Report on Task	RPROTS
TSO USERID	TSO User System Information	RTSUU(XXX)
TYPE OF JOB(STC BAT TSU)	CPU/TSO/MVS Overhead	RCPUT(XX)
TYPE OF JOB(STC TSO JOB)	Progress Report on Task	RPROTYPE
TYPE OF PAGE DATA SET	Storage/Paging/Slots	RASMT(XXX)
TYPE OF RESERVE	SYSPROG service RESERVE Interface	RREST(XXX)
TYPE OF UNIT	Tape or DASD Mount Requests	RMPTT(XXX)
TYPE/TASK ISSUING WTOR(STC TSU JOB)	SYSPROG service REPLIES Interface	RREPT(XXX)
TYPE/TASK ISSUING WTOR(STC TSU JOB)	SYSPROG service REPLIES Interface	RREPT.X
UCB OF DEVICE	SYSPROG service RESERVE Interface	RRESU(XXX)
UCB OF DEVICE	Interface to SYSPROG service TPIO	RTPIU(XXX)
UNIT ADDRESS	Find ALL Outstanding Non-TP I/O	RIOUA(XXX)
UNIT ADDRESS OF PAGE DATA SET	Storage/Paging/Slots	RASMU(XXX)
UNIT ADDRESS OF MOUNT	Tape or DASD Mount Requests	RMPTU(XXX)
VIO PAGING RATE	Paging/Demand Paging Information	RPAGVIO
VIO SLOTS	Storage/Paging/Slots	RASMUSPC
VIRTUAL USED ABOVE LINE	Progress Report on Task	RPROVUA
VIRTUAL USED BELOW LINE	Progress Report on Task	RPROVUB
VOLSER OF DEVICE	SYSPROG service RESERVE Interface	RRESV(XXX)
VOLSER OF DEVICE(MAY BE BLANK	Interface to SYSPROG service TPIO	RTPIV(XXX)
VOLSER OF REQUESTED VOLUME	Tape or DASD Mount Requests	RMPTV(XXX)
VOLUME SERIAL	Find ALL Outstanding Non-TP I/O	RIOVS(XXX)
VOLUME SERIAL	Determine I/O System Bottlenecks	RMDEV(XXX)
WORKING SET SIZE	Basic Real Storage Information	RRSMW(XXX)
WORKING SET SIZE	Return Status of Task(s)	RSTAW(XXX)

Glossary

This glossary defines BMC Software terminology. Other dictionaries and glossaries can be used in conjunction with this glossary.

Since this glossary pertains to BMC Software-related products, some of the terms defined might not appear in this book.

To help you find the information you need, this glossary uses the following cross-references:

Contrast with	Indicates a term that has a contrary or contradictory meaning.
See	Indicates an entry that is a synonym or contains expanded information.
See also	Indicates an entry that contains related information.

A

action. Defined operation, such as modifying a MAINVIEW window, that is performed in response to a command. *See* object.

active window. Any MAINVIEW window in which data can be refreshed. *See* alternate window, current window, window.

administrative view. Display from which a product's management tasks are performed, such as the DSLIST view for managing historical data sets. *See* view.

ALT WIN field. Input field that allows you to specify the window identifier for an alternate window where the results of a hyperlink are displayed. *See* alternate window.

Alternate Access. *See* MAINVIEW Alternate Access.

alternate form. View requested through the FORM command that changes the format of a previously displayed view to show related information. *See also* form, query.

alternate window. (1) Window that is specifically selected to display the results of a hyperlink. (2) Window whose identifier is defined to the ALT WIN field. *Contrast with* current window. *See* active window, window, ALT WIN field.

analyzer. (1) Online display that presents a snapshot of status and activity data and indicates problem areas. (2) Component of CMF MONITOR. *See* CMF MONITOR Analyzer.

application. (1) Program that performs a specific set of tasks within a MAINVIEW product. (2) In MAINVIEW VistaPoint, combination of workloads to enable display of their transaction performance data in a single view.

application trace. *See* trace.

ASCH workload. Workload comprising Advanced Program-to-Program Communication (APPC) address spaces.

AutoCustomization. Online facility for customizing the installation of products. AutoCustomization provides an ISPF panel interface that both presents customization steps in sequence and provides current status information about the progress of the installation.

automatic screen update. Usage mode wherein the currently displayed screen is refreshed automatically with new data at an interval you specify. Invoked by the ASU command.

B

batch workload. Workload consisting of address spaces running batch jobs.

BBI. Basic architecture that distributes work between workstations and multiple OS/390 targets for BMC Software MAINVIEW products.

BBI-SS PAS. *See* BBI subsystem product address space.

BBI subsystem product address space (BBI-SS PAS). OS/390 subsystem address space that manages communication between local and remote systems and that contains one or more of the following products:

- Command MQ for S/390
- MAINVIEW AutoOPERATOR
- MAINVIEW for CICS
- MAINVIEW for DB2
- MAINVIEW for DBCTL
- MAINVIEW for IMS Online
- MAINVIEW for MQSeries
- MAINVIEW SRM
- MAINVIEW VistaPoint (for CICS, DB2, DBCTL, and IMS workloads)

BBPARM. *See* parameter library.

BBPROC. *See* procedure library.

BBPROF. *See* profile library.

BBSAMP. *See* sample library.

BBV. *See* MAINVIEW Alternate Access.

BBXS. BMC Software Subsystem Services. Common set of service routines loaded into common storage and used by several BMC Software MAINVIEW products.

border. Visual indication of the boundaries of a window.

bottleneck analysis. Process of determining which resources have insufficient capacity to provide acceptable service levels and that therefore can cause performance problems.

C

CA-Disk. Data management system by Computer Associates that replaced the DMS product.

CAS. Coordinating address space. One of the address spaces used by the MAINVIEW windows environment architecture. The CAS supplies common services and enables communication between linked systems. Each OS/390 or z/OS image requires a separate CAS. Cross-system communication is established through the CAS using VTAM and XCF communication links.

CFMON. *See* coupling facility monitoring.

chart. Display format for graphical data. *See also* graph.

CICSplex. User-defined set of one or more CICS systems that are controlled and managed as a single functional entity.

CMF MONITOR. Comprehensive Management Facility MONITOR. Product that measures and reports on all critical system resources, such as CPU, channel, and device usage; memory, paging, and swapping activity; and workload performance.

CMF MONITOR Analyzer. Batch component of CMF MONITOR that reads the SMF user and 70 series records created by the CMF MONITOR Extractor and/or the RMF Extractor and formats them into printed system performance reports.

CMF MONITOR Extractor. Component of CMF that collects performance statistics for CMF MONITOR Analyzer, CMF MONITOR Online, MAINVIEW for OS/390, and RMF postprocessor. *See* CMF MONITOR Analyzer, CMF MONITOR Online, MAINVIEW for OS/390.

CMF MONITOR Online. Component of CMF that uses the MAINVIEW window interface to present data on all address spaces, their use of various system resources, and the delays that each address space incurs while waiting for access to these resources. *See* CMF MONITOR, MAINVIEW for OS/390.

CMF Type 79 API. Application programming interface, provided by CMF, that provides access to MAINVIEW SMF-type 79 records.

CMFMON. Component of CMF MONITOR that simplifies online retrieval of information about system hardware and application performance and creates MAINVIEW SMF-type 79 records.

The CMFMON *online facility* can be used to view data in one or more formatted screens.

The CMFMON *write facility* can be used to write collected data as MAINVIEW SMF-type 79 records to an SMF or sequential data set.

CMRDETL. MAINVIEW for CICS data set that stores detail transaction records (type 6E) and abend records (type 6D). Detail records are logged for each successful transaction. Abend records are written when an abend occurs. Both records have the same format when stored on CMRDETL.

CMRSTATS. MAINVIEW for CICS data set that stores both CICS operational statistic records, at five-minute intervals, and other records, at intervals defined by parameters specified during customization (using CMRSOPT).

column. Vertical component of a view or display, typically containing fields of the same type of information, that varies by the objects associated in each row.

collection interval. Length of time data is collected. *See also* delta mode, total mode.

command delimiter. Special character, usually a **;** (semicolon), used to stack commands typed concurrently on the COMMAND line for sequential execution.

COMMAND line. Line in the control area of the display screen where primary commands can be typed. *Contrast with* line command column.

Command MQ Automation D/S. Command MQ agents, which provide local proactive monitoring for both MQSeries and MSMQ (Microsoft message queue manager). The Command MQ agents operate at the local node level where they continue to perform functions regardless of the availability of the MQM (message queue manager) network. Functionality includes automatic monitoring and restarts of channels, queue managers, queues and command servers. In cases where automated recovery is not possible, the agents transport critical alert information to a central console.

Command MQ Automation S/390. Command MQ component, which monitors the MQM (message queue manager) networks and intercedes to perform corrective actions when problems arise. Solutions include:

- Dead-Letter Queue management
- System Queue Archival
- Service Interval Performance solutions
- Channel Availability

These solutions help ensure immediate relief to some of the most pressing MQM operations and performance problems.

Command MQ for D/S. Command MQ for D/S utilizes a true client/server architecture and employs resident agents to provide configuration, administration, performance monitoring and operations management for the MQM (message queue manager) network.

Command MQ for S/390. *See* MAINVIEW for MQSeries.

COMMON STORAGE MONITOR. Component of MAINVIEW for OS/390 that monitors usage and reconfigures OS/390 or z/OS common storage blocks.

composite workload. Workload made up of a WLM workload or other workloads, which are called *constituent workloads.*

constituent workload. Member of a composite workload. Constituent workloads in a composite usually belong to a single workload class, but sometimes are mixed.

contention. Occurs when there are more requests for service than there are servers available.

context. In a Plex Manager view, field that contains the name of a target or group of targets specified with the CONTEXT command. *See* scope, service point, SSI context, target context.

CONTEXT command. Specifies either a MAINVIEW product and a specific target for that product (*see* target context) or a MAINVIEW product and a name representing one or more targets (see *SSI context*) for that product.

control statement. (1) Statement that interrupts a sequence of instructions and transfers control to another part of the program. (2) Statement that names samplers and other parameters that configure the MAINVIEW components to perform specified functions. (3) In CMF MONITOR, statement in a parameter library member used to identify a sampler in the extractor or a report in the analyzer, or to describe either component's processing requirements to the operating system.

coupling facility monitoring (CFMON). Coupling facility views that monitor the activity of your system's coupling facilities.

current data. Data that reflects the system in its current state. The two types of current data are realtime data and interval data. *Contrast with* historical data. *See also* interval data and realtime data.

current window. In the MAINVIEW window environment, window where the main dialog with the application takes place. The current window is used as the default window destination for commands issued on the COMMAND line when no window number is specified. *Contrast with* alternate window. *See* active window, window.

D

DASD. Direct Access Storage Device. (1) A device with rotating recording surfaces that provides immediate access to stored data. (2) Any device that responds to a DASD program.

data collector. Program that belongs to a MAINVIEW product and that collects data from various sources and stores the data in records used by views. For example, MAINVIEW for OS/390 data collectors obtain data from OS/390 or z/OS services, OS/390 or z/OS control blocks, CMF MONITOR Extractor control blocks, and other sources. *Contrast with* extractor.

delta mode. (1) In MAINVIEW for DB2 analyzer displays, difference between the value sampled at the start of the current statistics interval and the value sampled by the current analyzer request. See also *statistics interval.* (2) In CMFMON, usage mode wherein certain columns of data reflect the difference in values between one sample cycle and the next. Invoked by the DELta ON command. *See also* collection interval, sample cycle, total mode.

DFSMS. Data Facility Storage Management System. Data management, backup, and HSM software from IBM for OS/390 or z/OS mainframes.

DMR. *See* MAINVIEW for DB2.

DMS. Data Management System. *See* CA-Disk.

DMS2HSM. *See* MAINVIEW SRM DMS2HSM.

DSO. Data Set Optimizer. CMF MONITOR Extractor component that uses CMF MONITOR Extractor data to produce reports specifying the optimal ordering of data sets on moveable head devices.

E

EasyHSM. *See* MAINVIEW SRM EasyHSM.

EasyPOOL. *See* MAINVIEW SRM EasyPOOL.

EasySMS. *See* MAINVIEW SRM EasySMS.

element. (1) Data component of a data collector record, shown in a view as a field. (2) Internal value of a field in a view, used in product functions.

element help. Online help for a field in a view. The preferred term is *field help.*

Enterprise Storage Automation. *See* MAINVIEW SRM Enterprise Storage Automation.

event. A message issued by Enterprise Storage Automation. User-defined storage occurrences generate events in the form of messages. These events provide an early warning system for storage problems and are routed to user-specified destinations for central viewing and management.

Event Collector. Component for MAINVIEW for IMS Online, MAINVIEW for IMS Offline, and MAINVIEW for DBCTL that collects data about events in the IMS environment. This data is required for Workload Monitor and optional for Workload Analyzer (except for the workload trace service). This data also is recorded as transaction records (X'FA') and program records (X'F9') on the IMS system log for later use by the MAINVIEW for IMS Offline components: Performance Reporter and Transaction Accountant.

expand. Predefined link from one display to a related display. *See also* hyperlink.

extractor. Program that collects data from various sources and keeps the data control blocks to be written as records. Extractors obtain data from services, control blocks, and other sources. *Contrast with* data collector.

extractor interval. *See* collection interval.

F

fast path. Predefined link between one screen and another. To use the fast path, place the cursor on a single value in a field and press Enter. The resulting screen displays more detailed information about the selected value. *See also* hyperlink.

field. Group of character positions within a screen or report used to type or display specific information.

field help. Online help describing the purpose or contents of a field on a screen. To display field help, place the cursor anywhere in a field and press PF1 (HELP). In some products, field help is accessible from the screen help that is displayed when you press PF1.

filter. Selection criteria used to limit the number of rows displayed in a view. Data that does not meet the selection criteria is not displayed. A filter is composed of an element, an operator, and an operand (a number or character string). Filters can be implemented in view customization, through the PARm/QPARm commands, or through the Where/QWhere commands. Filters are established against elements of data.

fire. The term used to indicate that an event has triggered an action. In MAINVIEW AutoOPERATOR, when a rule selection criteria matches an incoming event and *fires*, the user-specified automation actions are performed. This process is also called *handling* the event.

fixed field. Field that remains stationary at the left margin of a screen that is scrolled either right or left.

FOCAL POINT. MAINVIEW product that displays a summary of key performance indicators across systems, sites, and applications from a single terminal.

form. One of two constituent parts of a view; the other is query. A form defines how the data is presented; a query identifies the data required for the view. See also *query, view.*

full-screen mode. Display of a MAINVIEW product application or service on the entire screen. There is no window information line. *Contrast with* windows mode.

G

global command. Any MAINVIEW window interface command that can affect all windows in the window area of a MAINVIEW display.

graph. Graphical display of data that you select from a MAINVIEW window environment view. *See also* chart.

H

hilevel. For MAINVIEW products, high-level data set qualifier required by a site's naming conventions.

historical data. (1) Data that reflects the system as it existed at the end of a past recording interval or the duration of several intervals. (2) Any data stored in the historical database and retrieved using the TIME command. *Contrast with* current data, interval data and realtime data.

historical database. Collection of performance data written at the end of each installation-defined recording interval and containing up to 100 VSAM clusters. Data is extracted from the historical database with the TIME command. *See* historical data.

historical data set. In MAINVIEW products that display historical data, VSAM cluster file in which data is recorded at regular intervals.

HSM. (Hierarchical Storage Management) Automatic movement of files from hard disk to slower, less-expensive storage media. The typical hierarchy is from magnetic disk to optical disk to tape.

hyperlink. (1) Preset field in a view or an EXPAND line on a display that permits you to

- Access cursor-sensitive help
- Issue commands
- Link to another view or display

The transfer can be either within a single product or to a related display/view in a different BMC Software product. Generally, hyperlinked fields are highlighted. (2) Cursor-activated short path from a topic or term in online help to related information. *See also* fast path.

I

Image log. Collection of screen-display records. Image logs can be created for both the BBI-SS PAS and the BBI terminal session (TS).

The BBI-SS PAS Image log consists of two data sets that are used alternately: as one fills up, the other is used. Logging to the BBI-SS PAS Image log stops when both data sets are filled and the first data set is not processed by the archive program.

The TS Image log is a single data set that wraps around when full.

IMSPlex System Manager (IPSM). MVIMS Online and MVDBC service that provides Single System Image views of resources and bottlenecks for applications across one or more IMS regions and systems.

interval data. Cumulative data collected during a collection interval. Intervals usually last from 15 to 30 minutes depending on how the recording interval is specified during product customization. *Contrast with* historical data.

Note: If change is made to the workloads, a new interval will be started.

See also current data and realtime data.

InTune. Product for improving application program performance. It monitors the program and provides information used to reduce bottlenecks and delays.

IRUF. IMS Resource Utilization File (IRUF). IRUFs can be either detail (one event, one record) or summarized (more than one event, one record). A detail IRUF is created by processing the IMS system log through a program called IMFLEDIT. A summarized IRUF is created by processing one or more detail IRUFs, one or more summarized IRUFs, or a combination of both, through a sort program and the TASCOSTR program.

J

job activity view. Report about address space consumption of resources. *See* view.

journal. Special-purpose data set that stores the chronological records of operator and system actions.

Journal log. Collection of messages. Journal logs are created for both the BBI-SS PAS and the BBI terminal session (TS).

The BBI-SS PAS Journal log consists of two data sets that are used alternately: as one fills up, the other is used. Logging to the BBI-SS PAS Journal log stops when both data sets are filled and the first data set is not being processed by the archive program.

The TS Journal log is a single data set that wraps around when full.

L

line command. Command that you type in the line command column in a view or display. Line commands initiate actions that apply to the data displayed in that particular row.

line command column. Command input column on the left side of a view or display. *Contrast with* COMMAND line.

Log Edit. In the MAINVIEW for IMS Offline program named IMFLEDIT, function that extracts transaction (X‘FA’) and program (X‘F9’) records from the IMS system log. IMFLEDIT also extracts certain records that were recorded on the system log by IMS. IMFLEDIT then formats the records into a file called the IMS Resource Utilization File (IRUF).

M

MAINVIEW. BMC Software integrated systems management architecture.

MAINVIEW Alarm Manager. In conjunction with other MAINVIEW products, notifies you when an exception condition occurs. MAINVIEW Alarm Manager is capable of monitoring multiple systems simultaneously, which means that MAINVIEW Alarm Manager installed on one system keeps track of your entire sysplex. You can then display a single view that show exceptions for all MAINVIEW performance monitors within your OS/390 or z/OS enterprise.

MAINVIEW Alternate Access. Enables MAINVIEW products to be used without TSO by providing access through EXCP and VTAM interfaces.

MAINVIEW Application Program Interface. REXX- or CLIST-based, callable interface that allows MAINVIEW AutoOPERATOR EXECs to access MAINVIEW monitor product view data.

MAINVIEW AutoOPERATOR. Product that uses tools, techniques, and facilities to automate routine operator tasks and provide online performance monitoring, and that achieves high availability through error minimization, improved productivity, and problem prediction and prevention.

MAINVIEW control area. In the MAINVIEW window environment, first three lines at the top of the view containing the window information line and the COMMAND, SCROLL, CURR WIN, and ALT WIN lines. The control area cannot be customized and is part of the information display. *Contrast with* MAINVIEW display area, MAINVIEW window area.

MAINVIEW display area. *See* MAINVIEW window area.

MAINVIEW Explorer. Product that provides access to MAINVIEW products from a Web browser running under Windows. MAINVIEW Explorer replaces MAINVIEW Desktop.

MAINVIEW for CICS. Product (formerly MV MANAGER for CICS) that provides realtime application performance analysis and monitoring for CICS system management.

MAINVIEW for DB2. Product (formerly MV MANAGER for DB2) that provides realtime and historical application performance analysis and monitoring for DB2 subsystem management.

MAINVIEW for DBCTL. Product (formerly MV MANAGER for DBCTL) that provides realtime application performance analysis and monitoring for DBCTL management.

MAINVIEW for IMS (MVIMS) Offline. Product with a Performance Reporter component that organizes data and prints reports used to analyze IMS performance and a Transaction Accountant component that produces cost accounting and user charge-back records and reports.

MAINVIEW for IMS (MVIMS) Online. Product that provides realtime application performance analysis and monitoring for IMS management.

MAINVIEW for IP. Product that monitors OS/390 and z/OS mission-critical application performance as it relates to TCP/IP stack usage. Collected data includes availability, connections, response times, routers, service levels, storage, traffic, Web cache, and so on.

MAINVIEW for Linux–Servers. Product that allows you to monitor the performance of your Linux systems from the MAINVIEW windows interface.

MAINVIEW for MQSeries. Delivers comprehensive capabilities for configuration, administration, performance monitoring and operations management for an entire MQM (message queue manager) network.

MAINVIEW for OS/390. System management application (known as MAINVIEW **for MVS** prior to version 2.5). Built upon the MAINVIEW window environment architecture, it uses the window interface to provide access to system performance data and other functions necessary in the overall management of an enterprise.

MAINVIEW for UNIX System Services. System management application that allows you to monitor the performance of the Unix System Services from a MAINVIEW window interface.

MAINVIEW for VTAM. Product that displays application performance data by application, transaction ID, and LU name. This collected data includes: connections, response time statistics, application availability, and application throughput.

MAINVIEW for WebSphere. Product that provides Web monitoring and management for applications integrated with IBM WebSphere Application Server for OS/390 or z/OS.

MAINVIEW Selection Menu. ISPF selection panel that provides access to all MAINVIEW windows-mode and full-screen mode products.

MAINVIEW SRM. *See* MAINVIEW Storage Resource Manager (SRM).

MAINVIEW SRM DMS2HSM. Product that facilitates the conversion of CA-Disk, formerly known as DMS, to HSM.

MAINVIEW SRM EasyHSM. Product that provides online monitoring and reporting to help storage managers use DFHSM efficiently.

MAINVIEW SRM EasyPOOL. Product that provides control over data set allocation and enforcement of allocation and naming standards. EasyPOOL functions operate at the operating system level to intercept normal job processing, thus providing services without any JCL changes.

MAINVIEW SRM EasySMS. Product that provides tools that aid in the conversion to DFSMS and provides enhancement to the DFSMS environment after implementation. EasySMS consists of the EasyACS functions, the SMSACSTE function, and the Monitoring and Positioning Facility.

MAINVIEW SRM Enterprise Storage Automation. Product that delivers powerful event generation and storage automation technology across the storage enterprise. Used in conjunction with MAINVIEW AutoOPERATOR, automated solutions to perform pool, volume, application, or data set-level manipulation can be created and used in response to any condition or invoked to perform ad hoc requests

MAINVIEW SRM SG-Auto. Product that provides early warning notification of storage anomalies and automated responses to those anomalies based on conditions in the storage subsystem.

MAINVIEW SRM SG-Control. Product that provides real-time monitoring, budgeting, and control of DASD space utilization.

MAINVIEW SRM StopX37/II. Product that provides enhancements to OS/390 or z/OS space management, reducing the incidence of space-related processing problems. The StopX37/II functions operate at the system level to intercept abend conditions or standards violations, thus providing services without any JCL changes.

MAINVIEW SRM StorageGUARD. Product that monitors and reports on DASD consumption and provides historical views to help control current and future DASD usage.

MAINVIEW Storage Resource Manager (SRM). Suite of products that assists in all phases of OS/390 or z/OS storage management. MAINVIEW SRM consists of products that perform automation, reporting, trend analysis, and error correction for storage management.

MAINVIEW SYSPROG Services. *See* SYSPROG Services.

MAINVIEW VistaPoint. Product that provides enterprise-wide views of performance. Application and workload views are available for CICS, DB2, DBCTL, IMS, and OS/390. Data is summarized at the level of detail needed; for example, views can be for a single target, an OS/390 or z/OS image, or an entire enterprise.

MAINVIEW window area. Portion of the information display that is not the control area and in which views are displayed and windows opened. It includes all but the first three lines of the information display. *Contrast with* MAINVIEW control area.

monitor. Online service that measures resources or workloads at user-defined intervals and issues warnings when user-defined thresholds are exceeded.

Multi-Level Automation (MLA). The user-defined, multiple step process in Enterprise Storage Automation that implements solutions in a tiered approach, where solutions are invoked one after another until the condition is resolved.

MVALARM. *See* MAINVIEW Alarm Manager.

MVAPI. *See* MAINVIEW Application Program Interface.

MVCICS. *See* MAINVIEW for CICS.

MVDB2. *See* MAINVIEW for DB2.

MVDBC. *See* MAINVIEW for DBCTL.

MVIMS. *See* MAINVIEW for IMS.

MVIP. *See* MAINVIEW for IP.

MVLNX. *See* MAINVIEW for Linux–Servers.

MVMQ. *See* MAINVIEW for MQSeries.

MVMVS. *See* MAINVIEW for OS/390.

MVScope. MAINVIEW for OS/390 application that traces both CPU usage down to the CSECT level and I/O usage down to the channel program level.

MVSRM. *See* MAINVIEW Storage Resource Manager (SRM).

MVSRMHSM. *See* MAINVIEW SRM EasyHSM.

MVSRMSGC. *See* MAINVIEW SRM SG-Control.

MVSRMSGD. *See* MAINVIEW SRM StorageGUARD.

MVSRMSGP. *See* MAINVIEW SRM StorageGUARD.

MVVP. *See* MAINVIEW VistaPoint.

MVVTAM. *See* MAINVIEW for VTAM.

MVWEB. *See* MAINVIEW for WebSphere.

N

nested help. Multiple layers of help pop-up windows. Each successive layer is accessed by clicking a hyperlink from the previous layer.

O

object. Anything you can manipulate as a single unit. MAINVIEW objects can be any of the following: product, secondary window, view, row, column, or field.

You can issue an action against an object by issuing a line command in the line command column to the left of the object. *See* action.

OMVS workload. Workload consisting of OS/390 OpenEdition address spaces.

online help. Help information that is accessible online.

OS/390 and z/OS Installer. BMC Software common installation system for mainframe products.

OS/390 product address space (PAS). Address space containing OS/390 or z/OS data collectors, including the CMF MONITOR Extractor. Used by the MAINVIEW for OS/390, MAINVIEW for Unix System Services, and CMF MONITOR products. *See* PAS.

P

parameter library. Data set consisting of members that contain parameters for specific MAINVIEW products or a support component. There can be several versions:

- The distributed parameter library, called BBPARM
- A site-specific parameter library or libraries

  These can be

  – A library created by AutoCustomization, called UBBPARM

  – A library created manually, with a unique name

PAS. Product address space. Used by the MAINVIEW products. Contains data collectors and other product functions. *See* OS/390 product address space (PAS), BBI subsystem product address space (BBI-SS PAS).

performance group workload. Collection of address spaced defined to OS/390 or z/OS. If you are running OS/390 or z/OS with WLM in compatibility mode, MAINVIEW for OS/390 creates a performance group workload instead of a service class. *See* service class workload, workload definition.

PERFORMANCE MANAGER. MAINVIEW for CICS online service for monitoring and managing current performance of CICS regions.

Performance Reporter (MVIMS Offline). MVIMS Offline component that organizes data and prints reports that can be used to analyze IMS performance.

Performance Reporter. Product component that generates offline batch reports. The following products can generate these reports:

- MAINVIEW for DB2
- MAINVIEW for CICS

Plex Manager. Product through which cross-system communication, MAINVIEW security, and an SSI context are established and controlled. Plex Manager is shipped with MAINVIEW window environment products as part of the coordinating address space (CAS) and is accessible as a menu option from the MAINVIEW Selection Menu.

PRGP workload. In MVS/SP 5.0 or earlier, or in compatibility mode in MVS/SP 5.1 or later, composite of service classes. MAINVIEW for OS/390 creates a performance group workload for each performance group defined in the current IEAIPS*xx* member.

procedure library. Data set consisting of members that contain executable procedures used by MAINVIEW AutoOPERATOR. These procedures are execute command lists (EXECs) that automate site functions. There can be several versions:

- The distributed parameter library, called BBPROC
- A site-specific parameter library or libraries

  These can be

  – A library created by AutoCustomization, called UBBPROC

  – A library created manually, with a unique name

The site-created EXECs can be either user-written or customized MAINVIEW AutoOPERATOR-supplied EXECs from BBPROC.

product address space. *See* PAS.

profile library. Data set consisting of members that contain profile information and cycle refresh definitions for a terminal session connected to a BBI-SS PAS. Other members are dynamically created by MAINVIEW applications. There can be several versions:

- The distributed profile library, called BBPROF
- A site-specific profile library or libraries

  These can be

  – A library created by AutoCustomization, called SBBPROF

  – A library created manually, with a unique name

The site library is a common profile shared by all site users. The terminal session CLIST creates a user profile automatically if one does not exist; it is called userid.BBPROF, where userid is your logon ID. User profile libraries allow each user to specify unique PF keys, CYCLE commands, target system defaults, a Primary Option Menu, and a unique set of application profiles.

Q

query. One of two constituent parts of a view; the other is form. A query defines the data for a view; a form defines the display format. *See also* form, view.

R

realtime data. Performance data as it exists at the moment of inquiry. Realtime data is recorded during the smallest unit of time for data collection. *Contrast with* historical data. *See also* current data and interval data.

Resource Analyzer. Online realtime displays used to analyze IMS resources and determine which are affected by specific workload problems.

Resource Monitor. Online data collection services used to monitor IMS resources and issue warnings when defined utilization thresholds are exceeded.

row. (1) Horizontal component of a view or display comprising all the fields pertaining to a single device, address space, user, etc. (2) Horizontal component of a DB2 table consisting of a sequence of values, one for each column of the table.

RxD2. Product that provides access to DB2 from REXX. It provides tools to query the DB2 catalog, issue dynamic SQL, test DB2 applications, analyze EXPLAIN data, generate DDL or DB2 utility JCL, edit DB2 table spaces, perform security administration, and much more.

S

sample cycle. Time between data samples.

For the CMF MONITOR Extractor, this is the time specified in the extractor control statements (usually 1 to 5 seconds).

For realtime data, the cycle is not fixed. Data is sampled each time you press Enter.

sample library. Data set consisting of members each of which contains one of the following:

- Sample JCL that can be edited to perform specific functions
- A macro that is referenced in the assembly of user-written services
- A sample user exit routine

There can be several versions:

- The distributed sample library, called BBSAMP
- A site-specific sample library or libraries

  These can be

  – A library created by AutoCustomization, called UBBSAMP

  – A library created manually, with a unique name

sampler. Program that monitors a specific aspect of system performance. Includes utilization thresholds used by the Exception Monitor. The CMF MONITOR Extractor contains samplers.

SBBPROF. *See* profile library.

scope. Subset of an SSI context. The scope could be all the data for the context or a subset of data within the context. It is user- or site-defined. *See* SSI context, target.

screen definition. Configuration of one or more views that have been stored with the SAVEScr command and assigned a unique name. A screen includes the layout of the windows and the view, context, system, and product active in each window.

selection view. In MAINVIEW products, view displaying a list of available views.

service class workload. Collection of address spaces defined to OS/390 or z/OS. If you are running Workload Manager (WLM) in goal mode, MAINVIEW for OS/390 creates a service class workload for each service class that you define through WLM definition dialogs.

If you are running MVS 4.3 or earlier, or MVS/SP 5.1 or later with WLM in compatibility mode, MVS creates a performance group workload instead of a service class. *See* performance group workload.

service objective. Workload performance goal, specified in terms of response time for TSO workloads or turnaround time for batch workloads. Performance group workloads can be measured by either objective. Composite workload service objectives consist of user-defined weighting factors assigned to each constituent workload. For compatibility mode, neither OS/390 nor z/OS provides any way to measure service.

service point. Specification, to MAINVIEW, of the services required to enable a specific product. Services can be actions, selectors, or views. Each target (for example, CICS, DB2, or IMS) has its own service point.

The PLEX view lists all the defined service points known to the CAS to which the terminal session is connected.

service request block (SRB). Control block that represents a routine to be dispatched. SRB mode routines generally perform work for the operating system at a high priority. An SRB is similar to a task control block (TCB) in that it identifies a unit of work to the system. *See also* task control block.

service select code. Code entered to invoke analyzers, monitors, and general services. This code is also the name of the individual service.

session. Total period of time an address space has been active. A session begins when monitoring can be performed. If the product address space (PAS) starts after the job, the session starts with the PAS.

SG-Auto. *See* MAINVIEW SRM SG-Auto.

SG-Control. *See* MAINVIEW SRM SG-Control.

single system image (SSI). Feature of the MAINVIEW window environment architecture where you can view and perform actions on multiple OS/390 systems as though they were a single system. The rows of a single tabular view can contain rows from different OS/390 or z/OS images.

Skeleton Tailoring Facility. A facility in MAINVIEW AutoOPERATOR that allows skeleton JCL to be used during job submission. Skeleton JCL can contain variables within the JCL statements to be substituted with data values at job submission time. Directive statements can be used in the skeleton JCL to cause the repetition of a set of skeleton statements. This facility functions similar to the TSO skeleton tailoring facility.

SRB. *See* service request block.

SSI. *See* single system image.

SSI context. Name created to represent one or more targets for a given product. *See* context, target.

started task workload. Address spaces running jobs that were initiated programmatically.

statistics interval. For MAINVIEW for DB2, cumulative count within a predefined interval (30-minute default set by the DB2STATS parameter in the distributed BBPARM member BBIISP00) for an analyzer service DELTA or RATE display. Specifying the DELTA parameter displays the current value as the difference between the value sampled by the current analyzer request and the value sampled at the start of the current interval. Specifying the RATE parameter displays the current value by minute (DELTA divided by the number of elapsed minutes).

stem variables. A REXX facility, supported in MAINVIEW AutoOPERATOR REXX EXECs and the Skeleton Tailoring Facility, where variable names end with a period followed by a number, such as &POOL.1. This configuration allows each variable to actually represent a table or array of data, with the zero variable containing the number of entries in the array. For example, &POOL.0 = 5 would indicate variables &POOL.1 through &POOL.5 exist.

StopX37/II. *See* MAINVIEW SRM StopX37/II.

StorageGUARD. *See* MAINVIEW SRM StorageGUARD.

summary view. View created from a tabular view using the Summarize option in view customization. A summary view compresses several rows of data into a single row based on the summarize criteria.

SYSPROG services. Component of MAINVIEW for OS/390. Over 100 services that detect, diagnose, and correct OS/390 or z/OS system problems as they occur. Accessible from the OS/390 Performance and Control Main Menu. Note that this component is also available as a stand-alone product MAINVIEW SYSPROG Services.

system resource. *See* object.

T

target. Entity monitored by one or more MAINVIEW products, such as an OS/390 or z/OS image, an IMS or DB2 subsystem, a CICS region, or related workloads across systems. *See* context, scope, SSI context.

target context. Single target/product combination. *See* context.

TASCOSTR. MAINVIEW for IMS Offline program that summarizes detail and summary IMS Resource Utilization Files (IRUFs) to be used as input to the offline components.

task control block (TCB). Address space-specific control block that represents a unit of work that is dispatched in the address space in which it was created. *See also* service request block.

TCB. *See* task control block.

terminal session (TS). Single point of control for MAINVIEW products, allowing data manipulation and data display and providing other terminal user services for MAINVIEW products. The terminal session runs in a user address space (either a TSO address space or a standalone address space for EXCP/VTAM access).

TDIR. *See* trace log directory.

threshold. Specified value used to determine whether the data in a field meets specific criteria.

TLDS. *See* trace log data set.

total mode. Usage mode in CMFMON wherein certain columns of data reflect the cumulative value between collection intervals. Invoked by the DELta OFF command. *See also* collection interval, delta mode.

trace. (1) Record of a series of events chronologically listed as they occur. (2) Online data collection and display services that track transaction activity through DB2, IMS, or CICS.

trace log data set (TLDS). Single or multiple external VSAM data sets containing summary or detail trace data for later viewing or printing. The trace log(s) can be defined as needed or dynamically allocated by the BBI-SS PAS. Each trace request is assigned its own trace log data set(s).

trace log directory (TDIR). VSAM linear data set containing one entry for each trace log data set. Each entry indicates the date and time of data set creation, the current status of the data set, the trace target, and other related information.

transaction. Specific set of input data that initiates a predefined process or job.

Transaction Accountant. MVIMS Offline component that produces cost accounting and user charge-back records and reports.

TS. *See* terminal session.

TSO workload. Workload that consists of address spaces running TSO sessions.

U

UAS. *See* user address space.

UBBPARM. *See* parameter library.

UBBPROC. *See* procedure library.

UBBSAMP. *See* sample library.

user address space. Runs a MAINVIEW terminal session (TS) in TSO, VTAM, or EXCP mode.

User BBPROF. *See* profile library.

V

view. Formatted data within a MAINVIEW window, acquired from a product as a result of a view command or action. A view consists of two parts: query and form. *See also* form, job activity view, query.

view definition. Meaning of data that appears online, including source of data, selection criteria for data field inclusion and placement, data format, summarization, context, product, view name, hyperlink fields, and threshold conditions.

view command. Name of a view that you type on the COMMAND line to display that view.

view command stack. Internal stack of up to 10 queries. For each command, the stack contains the filter parameters, sort order, context, product, and timeframe that accompany the view.

view help. Online help describing the purpose of a view. To display view help, place the cursor on the view name on the window information line and press PF1 (HELP).

W

window. Area of the MAINVIEW screen in which views and resources are presented. A window has visible boundaries and can be smaller than or equal in size to the MAINVIEW window area. *See* active window, alternate window, current window, MAINVIEW window area.

window information line. Top border of a window. Shows the window identifier, the name of the view displayed in the window, the system, the scope, the product reflected by the window, and the timeframe for which the data in the window is relevant. *See also* window status field.

window number. Sequential number assigned by MAINVIEW to each window when it is opened. The window number is the second character in the window status field. *See also* window status field.

window status. One-character letter in the window status field that indicates when a window is ready to receive commands, is busy processing commands, is not to be updated, or contains no data. It also indicates when an error has occurred in a window. The window status is the first character in the window status field. *See also* window information line, window status field.

window status field. Field on the window information line that shows the current status and assigned number of the window. *See also* window number, window status.

windows mode. Display of one or more MAINVIEW product views on a screen that can be divided into a maximum of 20 windows. A window information line defines the top border of each window. *Contrast with* full-screen mode.

WLM workload. In goal mode in MVS/SP 5.1 and later, a composite of service classes. MAINVIEW for OS/390 creates a workload for each WLM workload defined in the active service policy.

workflow. Measure of system activity that indicates how efficiently system resources are serving the jobs in a workload.

workload. (1) Systematic grouping of units of work (e.g., address spaces, CICS transactions, IMS transactions) according to classification criteria established by a system administrator. (2) In OS/390 or z/OS, a group of service classes within a service definition.

workload activity view. Tracks workload activity as the workload accesses system resources. A workload activity view measures workload activity in terms of resource consumption and how well the workload activity meets its service objectives.

Workload Analyzer. Online data collection and display services used to analyze IMS workloads and determine problem causes.

workload definition. Workload created through the WKLIST view. Contains a unique name, a description, an initial status, a current status, and selection criteria by which address spaces are selected for inclusion in the workload. *See* Workload Definition Facility.

Workload Definition Facility. In MAINVIEW for OS/390, WKLIST view and its associated dialogs through which workloads are defined and service objectives set.

workload delay view. Tracks workload performance as the workload accesses system resources. A workload delay view measures any delay a workload experiences as it contends for those resources.

Workload Monitor. Online data collection services used to monitor IMS workloads and issue warnings when defined thresholds are exceeded.

workload objectives. Performance goals for a workload, defined in WKLIST. Objectives can include measures of performance such as response times and batch turnaround times.

Index

Symbols

A

B

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

STOP!

IMPORTANT INFORMATION - DO NOT INSTALL THIS PRODUCT UNLESS YOU HAVE READ ALL OF THE FOLLOWING MATERIAL

By clicking the YES or ACCEPT button below (when applicable), or by installing and using this Product or by having it installed and used on your behalf, You are taking affirmative action to signify that You are entering into a legal agreement and are agreeing to be bound by its terms, EVEN WITHOUT YOUR SIGNATURE. BMC is willing to license this Product to You ONLY if You are willing to accept all of these terms. CAREFULLY READ THIS AGREEMENT. If You DO NOT AGREE with its terms, DO NOT install or use this Product; press the NO or REJECT button below (when applicable) or promptly contact BMC or your BMC reseller and your money will be refunded if by such time You have already purchased a full-use License.

SOFTWARE LICENSE AGREEMENT FOR BMC PRODUCTS

SCOPE. This is a legally binding Software License Agreement (**"License"**) between You (either an individual or an entity) and BMC pertaining to the original computer files (including all computer programs and data stored in such files) contained in the enclosed Media (as defined below) or made accessible to You for electronic delivery, if as a prerequisite to such accessibility You are required to indicate your acceptance of the terms of this License, and all whole or partial copies thereof, including modified copies and portions merged into other programs (collectively, the **"Software"**). **"Documentation"** means the related hard-copy or electronically reproducible technical documents furnished in association with the Software, **"Media"** means the original BMC-supplied physical materials (if any) containing the Software and/or Documentation, **"Product"** means collectively the Media, Software, and Documentation, and all Product updates subsequently provided to You, and **"You"** means the owner or lessee of the hardware on which the Software is installed and/or used. **"BMC"** means BMC Software Distribution, Inc. unless You are located in one of the following regions, in which case "BMC" refers to the following indicated BMC Software, Inc. subsidiary: (i) Europe, Middle East or Africa --BMC Software Distribution, B.V., (ii) Asia/Pacific -- BMC Software Asia Pacific Pte Ltd., (iii) Brazil -- BMC Software do Brazil, or (iv) Japan -- BMC Software K.K. **If You enter into a separate, written software license agreement signed by both You and BMC or your authorized BMC reseller granting to you the rights to install and use this Product, then the terms of that separate, signed agreement will apply and this License is void.**

FULL-USE LICENSE. Subject to these terms and payment of the applicable license fees, BMC grants You this non-exclusive License to install and use one copy of the Software for your internal use on the number(s) and type(s) of servers or workstations for which You have paid or agreed to pay to BMC or your BMC reseller the appropriate license fee. If your license fee entitles You only to a License having a limited term, then the duration of this License is limited to that term; otherwise this License is perpetual, subject to the termination provisions below.

TRIAL LICENSE. If You have not paid or agreed to pay to BMC or your BMC Reseller the appropriate license fees for a full use license, then, **NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENSE**: (i) this License consists of a non-exclusive evaluation license ("Trial License") to use the Product for a limited time ("Trial Period") only for evaluation; (ii) during the Trial Period, You may not use the Software for development, commercial, production, database management or other purposes than those expressly permitted in clause (i) immediately above; and (iii) your use of the Product is on an ***AS IS*** basis, and **BMC, ITS RESELLERS AND LICENSORS GRANT NO WARRANTIES OR CONDITIONS (INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE) TO YOU AND ACCEPT NO LIABILITY WHATSOEVER RESULTING FROM THE USE OF THIS PRODUCT UNDER THIS TRIAL LICENSE.** If You use this Product for other than evaluation purposes or wish to continue using it after the Trial Period, you must purchase a full-use license. When the Trial Period ends, your right to use this Product automatically expires, though in certain cases You may be able to extend the term of the Trial Period by request. Contact BMC or your BMC reseller for details.

TERM AND TERMINATION. This License takes effect on the first to occur of the date of shipment or accessibility to You for electronic delivery, as applicable (the **"Product Effective Date"**). You may terminate this License at any time for any reason by written notice to BMC or your BMC reseller. This License and your right to use the Product will terminate automatically with or without notice by BMC if You fail to comply with any material term of this License. Upon termination, You must erase or destroy all components of the Product including all copies of the Software, and stop using or accessing the Software. Provisions concerning Title and Copyright, Restrictions (or Restricted Rights, if You are a U.S. Government entity) or limiting BMC's liability or responsibility shall survive any such termination.

TITLE AND COPYRIGHT; RESTRICTIONS. All title and copyrights in and to the Product, including but not limited to all modifications thereto, are owned by BMC and/or its affiliates and licensors, and are protected by both United States copyright law and applicable international copyright treaties. You will not claim or assert title to or ownership of the Product. To the extent expressly permitted by applicable law or treaty notwithstanding this limitation, You may copy the Software only for backup or archival purposes, or as an essential step in utilizing the Software, but for no other purpose. You will not remove or alter any copyright or proprietary notice from copies of the Product. You acknowledge that the Product contains valuable trade secrets of BMC and/or its affiliates and licensors. Except in accordance with the terms of this License, You agree (a) not to decompile, disassemble, reverse engineer or otherwise attempt to derive the Software's source code from object code except to the extent expressly permitted by applicable law or treaty despite this limitation; (b) not to sell, rent, lease, license, sublicense, display, modify, time share, outsource or otherwise transfer the Product to, or permit the use of this Product by, any third party; and (c) to use reasonable care and protection to prevent the unauthorized use, copying, publication or dissemination of the Product and BMC confidential information learned from your use of the Product. **You will not export or re-export any Product without both the written consent of BMC and the appropriate U.S. and/ or foreign government license(s) or license exception(s).** Any programs, utilities, modules or other software or documentation created, developed, modified or enhanced by or for You using this Product shall likewise be subject to these restrictions. BMC has the right to obtain injunctive relief against any actual or threatened violation of these restrictions, in addition to any other available remedies. Additional restrictions may apply to certain files, programs or data supplied by third parties and embedded in the Product; consult the Product installation instructions or Release Notes for details.

LIMITED WARRANTY AND CONDITION. If You have purchased a Full-Use License, BMC warrants that (i) the Media will be, under normal use, free from physical defects, and (ii) for a period of ninety (90) days from the Product Effective Date, the Product will perform in substantial accordance with the operating specifications contained in the Documentation that is most current at the Product Effective Date. BMC's entire liability and your exclusive remedy under this provision will be for BMC to use reasonable best efforts to remedy defects covered by this warranty

and condition within a reasonable period of time or, at BMC's option, either to replace the defective Product or to refund the amount paid by You to license the use of the Product. BMC and its suppliers do not warrant that the Product will satisfy your requirements, that the operation of the Product will be uninterrupted or error free, or that all software defects can be corrected. This warranty and condition shall not apply if: (i) the Product is not used in accordance with BMC's instructions, (ii) a Product defect has been caused by any of your or a third party's malfunctioning equipment, (iii) any other cause within your control causes the Product to malfunction, or (iv) You have made modifications to the Product not expressly authorized in writing by BMC. No employee, agent or representative of BMC has authority to bind BMC to any oral representations, warranties or conditions concerning the Product. **THIS WARRANTY AND CONDITION IS IN LIEU OF ALL OTHER WARRANTIES AND CONDITIONS. THERE ARE NO OTHER EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS, INCLUDING THOSE OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, REGARDING THIS LICENSE OR ANY PRODUCT LICENSED HEREUNDER. THIS PARAGRAPH SHALL NOT APPLY TO A TRIAL LICENSE.** Additional support and maintenance may be available for an additional charge; contact BMC or your BMC reseller for details.

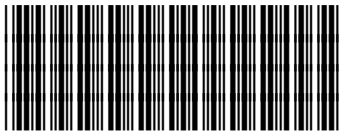
LIMITATION OF LIABILITY. Except as stated in the next succeeding paragraph, BMC's and your BMC reseller's total liability for all damages in connection with this License is limited to the price paid for the License. **IN NO EVENT SHALL BMC BE LIABLE FOR ANY CONSEQUENTIAL, SPECIAL, INCIDENTAL, PUNITIVE OR INDIRECT DAMAGES OF ANY KIND ARISING OUT OF THE USE OF THIS PRODUCT (SUCH AS LOSS OF PROFITS, GOODWILL, BUSINESS, DATA OR COMPUTER TIME, OR THE COSTS OF RECREATING LOST DATA), EVEN IF BMC HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Some jurisdictions do not permit the limitation of consequential damages so the above limitation may not apply.

INDEMNIFICATION FOR INFRINGEMENT. BMC will defend or settle, at its own expense, any claim against You by a third party asserting that your use of the Product within the scope of this License violates such third party's patent, copyright, trademark, trade secret or other proprietary rights, and will indemnify You against any damages finally awarded against You arising out of such claim. However, You must promptly notify BMC in writing after first receiving notice of any such claim, and BMC will have sole control of the defense of any action and all negotiations for its settlement or compromise, with your reasonable assistance. BMC will not be liable for any costs or expenditures incurred by You without BMC's prior written consent. If an order is obtained against your use of the Product by reason of any claimed infringement, or if in BMC's opinion the Product is likely to become the subject of such a claim, BMC will at its option and expense either (i) procure for You the right to continue using the product, or (ii) modify or replace the Product with a compatible, functionally equivalent, non-infringing Product, or (iii) if neither (i) nor (ii) is practicable, issue to You a pro-rata refund of your paid license fee(s) proportionate to the number of months remaining in the 36 month period following the Product Effective Date. This paragraph sets forth your only remedies and the total liability to You of BMC, its resellers and licensors arising out of such claims.

GENERAL. This License is the entire understanding between You and BMC concerning this License and may be modified only in a mutually signed writing between You and BMC. If any part of it is invalid or unenforceable, that part will be construed, limited, modified, or, severed so as to eliminate its invalidity or unenforceability. This License will be governed by and interpreted under the laws of the jurisdiction named below, without regard to conflicts of law principles, depending on which BMC Software, Inc. subsidiary is the party to this License: (i) BMC Software Distribution, Inc. - the State of Texas, U.S.A., (ii) BMC Software Distribution, B.V. - The Netherlands, (iii) BMC Software Asia Pacific Pte Ltd. -- Singapore (iv) BMC Software do Brazil -- Brazil, or (v) BMC Software K.K. -- Japan. Any person who accepts or signs changes to the terms of this License promises that they have read and understood these terms, that they have the authority to accept on your behalf and legally obligate You to this License. Under local law and treaties, the restrictions and limitations of this License may not apply to You; You may have other rights and remedies, and be subject to other restrictions and limitations.

U.S. GOVERNMENT RESTRICTED RIGHTS. UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT LAWS OF THE UNITED STATES. Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in FAR Section 52.227-14 Alt. III (g)(3), FAR Section 52.227-19, DFARS 252.227-7014 (b) or DFARS 227.7202, as amended from time to time. Contractor/Manufacturer is BMC Software, Inc., 2101 CityWest Blvd., Houston, TX 77042-2827, USA. Any contract notices should be sent to this address.

Notes



100042409